A denotational theory of synchronous communicating systems

Albert Benveniste, Paul Le Guernic

INRIA

UNITÉ DE RECHERCHE
INRIA-RENNES

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P.105
78153 Le Chesnay Cedex
France

Tél. (1) 39 63 55 11

Rapports de Recherche

N° 685

A DENOTATIONAL THEORY OF SYNCHRONOUS COMMUNICATING SYSTEMS

Albert BENVENISTE
Paul LE GUERNIC

JUIN 1987

A DENOTATIONAL THEORY

OF SYNCHRONOUS COMMUNICATING SYSTEMS.

Albert BENVENISTE, Paul LE GUERNIC

IRISA/INRIA
Campus de Beaulieu
35042 RENNES CEDEX, FRANCE

Abstract: *Synchronous systems are considered as functions transforming histories. To properly deal with the temporal dependencies between actions, the notions of event and signal are reformulated as functions of the history of external stimuli. The algebra of all the events and signals which can be built on a given history is completely described. Models of communicating histories and processes are introduced, which reveal how nondeterminism and starvation can still occur under the assumption of synchrony. It is shown how the calculus of the synchronisation of a process is represented by a suitable dynamical system over the finite field of the integers modulo 3. Relations of this model with synchronous languages such as ESTEREL, LUSTRE, SIGNAL, is discussed.*

UNE THEORIE DENOTATIONNELLE

DES PROCESSUS SYNCHRONES COMMUNICANTS.

Résumé: *On considère les systèmes synchrones comme transformateurs d'histoires. Pour tenir compte des dépendances temporelles entre actions, les notions d'évènement et de signal sont considérées comme fonctions des stimuli externes. L'algèbre de tous les évènements et signaux qui peuvent être construits sur une histoire donnée est complètement décrite. On introduit des modèles décrivant la communication entre histoires et processus, ces modèles permettent de comprendre pourquoi famine et non-déterminisme peuvent se rencontrer dans les systèmes synchrones. On exhibe une algèbre de systèmes dynamiques sur le corps fini des entiers modulo 3, qui résume complètement la synchronisation d'un processus. Finalement, on présente les liens entre ce modèle et les langages synchrones ESTEREL, LUSTRE, et SIGNAL.*

Chapter One
INTRODUCTION

According to [Young 1982], a real time system is *any information processing activity or system which has to respond to externally generated stimuli within a finite and specifiable delay*. A further feature of ***synchronous*** real time systems is that to every event can be assigned a time at which this event occurs. A consequence is that, to avoid any non determinism, the set of the external stimuli has to be fixed and specified in advance. Since our aim is to study the notion of event in real time systems, we shall consider events as *functions of the input stimuli*; this will allow us to take into account the temporal causality constraints in a natural framework.

This point of view about synchrony, serves as a basis for the recent development of three *synchronous languages*: the imperative language ESTEREL ([Berry & Cosserat 1984], [Boussinot 1986]), and the data−flow oriented languages LUSTRE ([Bergerand & al. 1985]) and SIGNAL ([Le Guernic & al. 1986], [Le Guernic & Benveniste 1986]).

Finally, we should say that the ideas underlying the present approach go back to the classical point of view of the probability and ergodic theories to handle the time ([Dellacherie & Meyer 1976]).

1.1 What is the «time» in real−time synchronous systems?

While classical (i.e. asynchronous) real time languages do implicitely or explicitely refer to some external and universal time reference, the notion of «time» is completely different in synchronous real time systems. To be more explicit, synchronous real time systems differ from asynchronous ones in the following aspects:

1. *concerning the internal mechanisms of the system:* every action is instantaneous, i.e. has a zero duration;
2. *concerning the communications with the external world:* the set of the possible input stimuli is fixed and known in advance, and input flows are specified through both
   - the values they carry
   - a total ordering of the «instants» at which these values are available at the external ports.

Of course, this last requirement is the fundamental feature which characterizes the way synchronous real time systems communicate with the external world, compared to asynchronous ones. Let us illustrate further this point on a simple example.

Consider a real time system with two inputs:

1. a data input carrying an ordered file of data named x,
2. an interrupt input port named s.

Then, the specification of an input history according to the synchronous point of view must be of the form

$$\begin{array}{llll} x_1 & x_2 & x_3 & \perp \\ \perp, & s_2, & \perp, & s_4, \ etc... \end{array}$$

(as usually, $\perp$ denotes here the absence of data) i.e. both the values and their global ordering must be specified; the integer index $t = 1, 2, ...$ is used for this purpose. And *this index "t" has to be considered as the proper notion of time in synchronous systems.* Another fundamental consequence is that the notion of time is *local to a given process*: there is no universal time reference, as we shall see later when process interconnections will be studied.

In other words, the essentially nondeterministic character of the communications with the external world in real time systems is concentrated here inside some (ignored) external mechanism which *decides* this global ordering. Hence, the advantage of the synchronous point of view is that the nondeterminism of externall communications is strictly concentrated in this external mechanism, and is thus by no means propagated inside the body of the system itself. For this reason, G. Berry oftens refers to synchronous real time system as *reactive systems*, i.e. systems which react instantaneously to external input stimuli in a functional way, although this point of view has to be slightly refined as we shall see later.

1.2 What is the proper notion of «process» in synchronous systems?

It is usual to consider that processes which have the same external behaviour are indistinguishable; usual descriptions of external behaviours in concurrent systems are, for example, the equivalence classes of strong and weak bisimulation in [Milner 1982], and the pairs {traces, failure set} used in CSP ([Brookes & al. 1984]).

In our case however, the notion of external behaviour can be defined in a richer way, thanks to the synchrony assumption. Let us discuss informally this point. Consider a synchronous real time system, and denote by

$$\begin{array}{l} X_t = (x_t(1), ..., x_t(n)) \\ Y_t = (y_t(1), ..., y_t(p)) \end{array} \qquad (1\text{-}1)$$

the set of its input stimuli and outputs respectively, where the index *"t"* refers to the time index. Denote by

$$X_{[1,t]} = (X_1, ..., X_t) \qquad (1\text{-}2)$$

Table of Contents

1. INTRODUCTION 1
   1.1 What is the «time» in real−time synchronous systems? 1
   1.2 What is the proper notion of «process» in synchronous systems? 2
   1.3 What is the proper notion of «event» in real time synchronous systems? 3
   1.4 About the style of the presented model. 3
   1.5 Organization of the paper. 4
   1.6 A basic example: the family of the mini-languages SLi:i=0,...,4 5
      1.6.1 The instructions. 5
      1.6.2 The languages. 6
      1.6.3 A simple programming example in SL4: the time multiplexing. 6

2. THE MODEL OF HISTORIES. 9
   2.1 Basic notions: histories, signals, clocks. 9
      2.1.1 Histories. 9
         2.1.1.1 Notations. 9
         2.1.1.2 Two simple examples. 10
      2.1.2 Clocks. 10
         2.1.2.1 Definition of clocks. 11
         2.1.2.2 Counters. 11
         2.1.2.3 Traces. 12
         2.1.2.4 Examples and counterexamples. 12
         2.1.2.5 History associated to a clock. 14
      2.1.3 Causal signals. 14
   2.2 The clock algebra. 15
      2.2.1 A partial order on the set of the clocks. 15
      2.2.2 The filtering. 16
      2.2.3 The multiplexing. 17
      2.2.4 Synchronizable clocks. 20
   2.3 History communication and time changes. 21
      2.3.1 Examples. 21
      2.3.2 History juxtaposition. 21
         2.3.2.1 STEP 1: inserting dummy events. 21
         2.3.2.2 STEP 2: defining history juxtaposition. 23
      2.3.3 History communication. 23
         2.3.3.1 Definition. 24
         2.3.3.2 Examples again. 24
      2.3.4 Embeddings. 26
      2.3.5 Determinism. 27
         2.3.5.1 Example pursued. 27
         2.3.5.2 Temporal nondeterminism, and observer's viewpoint. 27

2.4 History congruence. 28
2.4.1 Isomorphisms. 28
2.4.2 Congruence. 28
2.5 Summary of the chapter, and discussion. 29

3. THE MODEL OF PROCESSES. 31
3.1 Processes. 31
3.1.1 Definition of processes. 31
3.1.2 History generated by a family of ports. 31
3.2 Process communication. 32
3.3 Process congruence. 33
3.4 The clock calculus. 33
3.4.1 Process synchronization. 34
3.4.2 Definition of the clock calculus. 34
3.5 The algebraic clock calculus. 35
3.5.1 Preliminaries. 35
3.5.2 The fundamental map. 37
3.5.3 Starvation, temporal nondeterminism, and clock calculus. 38
3.5.3.1 Starvation and death time. 38
3.5.3.2 Temporal nondeterminism. 39

4. A DENOTATIONAL SEMANTICS FOR THE *SL* LANGUAGES. 41
4.1 Notations. 41
4.1.1 The semantic function. 41
4.1.2 Notations for the field «EPROC» 42
4.1.2.1 Canonical process associated to a signal. 42
4.1.2.2 Canonical process associated to a variable. 42
4.1.2.3 Use of process communication. 42
4.1.3 Notations for the field «SYNCH». 43
4.1.4 Notations for the field «VAL». 43
4.1.5 Notations for the field «ALGCLOCK». 44
4.2 The semantics of *SL*. 44
4.2.1 Instruction (o): relation. 45
4.2.1.1 Non boolean relation. 45
4.2.1.2 Boolean relation. 45
4.2.2 Instruction (i): the register. 46
4.2.2.1 Non boolean register. 46
4.2.2.2 Boolean register. 46
4.2.3 Instruction (ii): the condition. 47
4.2.3.1 Condition with non boolean output. 47
4.2.3.2 Condition with boolean output. 47
4.2.4 Instruction (iii): the multiplexing. 48
4.2.5 Instruction (iv): the merge. 48

4.2.5.1 Merge with non boolean output. 48
4.2.5.2 Merge with boolean output. 48
4.2.6 Instruction (v): communication. 49
4.3 Applications. 49
4.3.1 Proof that SL3 is included in SL4. 49
4.3.2 The algebraic clock calculus as a proof system. 52
4.3.2.1 Determinism. 53
4.3.2.2 Starvation. 54
4.3.2.3 Some hints for effective algorithms. 55
4.4 Conclusion of this chapter. 55

5. CONCLUSION AND DISCUSSION. 57
5.1 Conclusion: W-calculus. 57
5.2 About the synchronous languages. 57
5.3 Future directions of research. 58

the *initial segment* up to time t, with the corresponding notation for the output Y. Then, a natural definition of the external behaviour of a process is the set of its *histories*, defined informally as *the collection of input/output initial segments* $\{X_{[1,t]}, Y_{[1,t]}\}_{t \in T}$, where T denotes the time index set. Consequently, we follow a point of view close to [Kahn 1974], [Kahn & Mc Queen 1977], by considering that processes are *the collection of input-output maps which relate input initial segments to output initial segments*; as usually, fixed point arguments can be used to defined a global input-output map, the restrictions of which are consistent with the previously given collection. To insist on the fact that processes do relate initial segments, we shall throughout the introduction use the name of ***causal processes.***

1.3 What is the proper notion of «event» in real time synchronous systems?

Real time synchronous systems are closed worlds, since they do communicate with the external world through a fixed set of stimuli. As a consequence, the point of view we have taken to define causal processes should be also followed for defining events: events must be considered as *functions of the stimuli* satisfying a suitable counterpart of the causality condition we have introduced for processes. Such a condition is somewhat more involved, and will be introduced later. Events which are correct in the above (sketchy) mentioned sense will be called ***clocks.*** Clocks are definitely the most subtle objects to take place in synchronous real time systems. Basic problems related to clocks are the following:

- how to characterize clocks in such a way that the so defined class does include the class of the events which are produced by the monitoring of causal processes?
- as one can guess, clocks can be used to generate *time changes*, and new causal processes and clocks can be generated subjected to this new timing; this is a good way to consider that time is a local rather than universal notion; but how to guaranty that such a procedure will preserve the causality with respect to the original input stimuli?
- what are the convenient primitive operations on the time to be able to build any new time reference from the original one?
- how process communication is related to time changes?

1.4 About the style of the presented model.

There are mainly two classes of styles for models of concurrent processes: the *operational* models and the *denotational* ones.

In the operational style advocated in [Plotkin 1981], processes are dynamic objects which evolve under the action of events in a way which is specified by rules. The operational style makes the study of communication and concurrency of dynamic processes quite easy, and leads very quickly to effective implementations. To the operational family belong for example SCCS of

[Milner 1982], the operational semantics of the synchronous languages ESTEREL [Berry & Cosserat 1984] and SIGNAL [Le Guernic & Benveniste 1986], and also to some extend the failure based CSP model of [Brookes & al. 1984], and the work of Cardelli on analog processes and real time agents ([Cardelli 1980, 1982]).

To our knowledge, the pioneering work relevant to denotational style is the Dynamic Network Processes model introduced in [Kahn 1974]. To this family belong also the deep study [De Bruin & Boehm 1985] on Kahn's approach. The advantage of this approach is its elegance, but a severe drawback is the necessity to rely on a difficult *continuation* technique ([De Bruin & Boehm 1985]) to study the communication of processes. Our approach belongs also to this family, so that it enjoys the same advantages, but to some extend suffers from the same drawbacks; however, it is our opinion that the fully functional viewpoint of processes and clocks we have followed allows us to cope with the use of the continuation technique.

1.5 Organization of the paper.

The paper is organized as follows.

In the end of the introduction, a family of mini-languages SL_i (SL for «synchronous language») is presented, which will illustrate our notions throughout this paper.

The second chapter is devoted to the model of histories. This is a very abstract model of synchronous communicating systems. Its interest lies in the fact that almost all the difficulties of the subject are present, although this model is fairly general. A first section concentrates on the basic notions of this model: histories, signals, and clocks. The algebra of all the clocks of a given history is analysed in the second section; the theorem 2 is a fundamental one, it asserts that the unusual operation of time-multiplexing of events is necessary for a model of synchronous communicating systems to be sufficiently rich. Then, the last section of this chapter is devoted to the presentation of history communication, which is the core of the subject.

Since the latter notion was introduced in a rather abstract setting, a more concrete point of view is taken in the third chapter, where the model of processes is introduced, by just expressing that processes are histories which do communicate via visible ports only. It is shown how synchronisation actions can be summed up and calculated with the aid of the «clock calculus».

Finally, an illustration of the previous models on the SL_i mini-languages is presented in the last chapter.

1.6 A basic example: the family of the mini-languages $SL_i : i = 0, \ldots, 4$

Here, *SL* refers to «synchronous language». These languages will be useful to illustrate our purpose, and will allow us to propose a simple classification of existing languages relevant to the synchronous approach.

1.6.1 The instructions.

Consider the following instructions:

p(x(1),...,x(n))	(o)
y = u → $x	(i)
y = x **when** b	(ii)
h = **mux** c	(iii)
y = u **default** v	(iv)
P ‖ Q	(v)

Their intuitive meaning is the following:

(o): direct extension of instantaneous relations into relations acting on flows

$$p(x(1),\ldots,x(n)) \Leftrightarrow \forall t : p(x_t(1),\ldots,x_t(n))$$

An example of relation is a function y = g(x(1),...,x(n)), or a logical assertion such as b **and not** c.

(i): register:

$$y = u \rightarrow \$x \Leftrightarrow \forall t > 1 : y_t = x_{t-1}, y_1 = u$$

(ii): condition (b is boolean): y equals x when the signal x and the boolean b are available and b is true;

(iii): multiplexing (c is integer): when c is received, and carries the integer value *n*, the pure signal (say a boolean always true) h is emitted simultaneously with c, and then successively *n* times immediately after c has been received; then, the instruction is ready to receive c again; this is the basic instruction to produce oversampling;

(iv): y merges u and v, with priority to u when both signals are simultaneously present;

The instructions (o−iv) specify the elementary processes, we shall call *generators*. The objects named x, y, u, v, b, will be called *signals*.

(v): communication of already defined processes: P and Q communicate through their signals with common names; for example

$$y = zy + a \;\|\; zy = u \rightarrow \$y$$

denotes the system of recurrent equations for $t \geqq 1$

$$y_t = zy_t + a_t$$
$$zy_t = y_{t-1}, zy_1 = u$$

which is equivalent to $y_t = y_{t-1} + a_t, y_0 = u$.

1.6.2 The languages.

The family of languages SL_i is then defined as follows

- SL_0: instructions (o),(v)
- SL_1: instructions (o),(i),(v)
- SL_2: instructions (o),(i),(ii),(v)
- SL_3: instructions (o),(i),(ii),(iii),(v)
- SL_4: instructions (o),(i),(ii),(iv),(v)

We shall make the following uses of the family SL_i. First, we shall use it to illustrate our model. Second, this family exhibits easily the following classification: $SL_0 \subset SL_1 \subset SL_2 \subset SL_3 \subset SL_4$; the last inclusion will be shown later. Finally, we shall see that there is a strong relation respectively between SL_1 and LUCID ([Ashcroft & Wadge 1976]), SL_2 and LUSTRE, SL_3 and ESTEREL, and finally SL_4 and SIGNAL. This will allow us to discuss the respective advantages and drawbacks of these languages.

1.6.3 A simple programming example in SL_4: the time multiplexing.

Let C denote an integer valued signal; consider the following SL_4 program

(i1) u = zu − 1
‖
(i2) zu = raz **default** past_u
‖
(i3) past_u = u_0 → $u

‖

```
(i4)    raz = C when past_stop
        ||
(i5)    past_stop = true → $stop
        ||
(i6)    stop = (u = 0)

        ||

(i7)    synchro C, raz
```

Let us analyse this program.
The instructions (i1)−(i3) define a decreasing counter with a reset signal denoted by "raz". The instruction (i1) is a function, i.e. an instruction of type (o), with zu as input signal, and u as output signal. The instructions (i2)(i3) indicate that the signal zu is the past value of u, except when the reset signal raz is received.
The instructions (i4)−(i6) define a subprocess having the signal raz as output. On the other hand, the instructions (i5)(i6) expresse that raz is delivered at the next instant after u=0.
Finally, **synchro** denotes the function (instruction of type (o)) with inputs only, so that its effect is only to express that the signals C and raz must be simultaneously available.

The behaviour of this program is informally depicted in the following figure, where the upward arrows correspond to a decrease of the counter u.

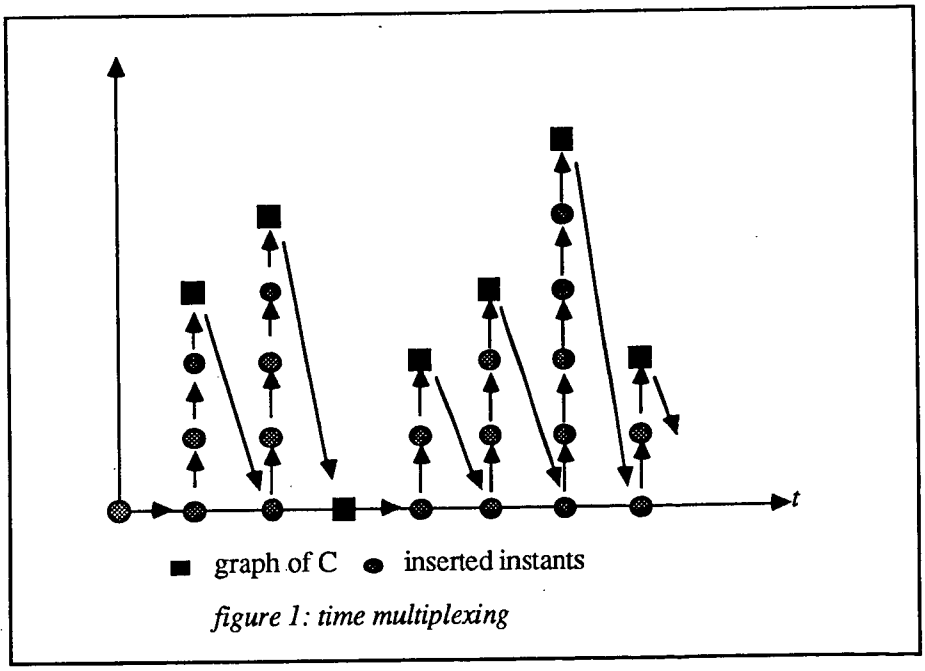


figure 1: time multiplexing

It is clear that, if h denotes a pure signal simultaneously available with u, then h and C are related through the instruction (iii), namely h = **mux** C. Notice also that this program possesses "C" as only input, whereas its output "u" has a faster clock than the clock of the input C. This informal discussion will serve as a motivation for the model we shall introduce now, and we shall re-use some of the instructions (i1-7) to illustrate various objects of this model.

Chapter Two
THE MODEL OF HISTORIES.

2.1 Basic notions: histories, signals, clocks.

2.1.1 Histories.

In the sequel, the symbols $\mathbf{N}, \mathbf{N}_+$ denote respectively the ordered sets $\{0,1,2,\ldots,\infty\}$ and $\{1,2,\ldots,\infty\}$. By ∞, we have in mind an instant for which one waits for ever, i.e. an event occurring at ∞ never happens.

Definition 1: *By a* ***history****, we have in mind an object*

$$\{\Omega,(\Pi_t)_{t\in\mathbf{N}}\} \quad \text{or} \quad \{\Omega,\Pi\} \quad \text{for short}$$

where

- Ω *is a set*
- *for every* t, Π_t *is a partition of the set* Ω, *the elements of which are referred to as* ***atoms***
- *for* $s<t$, Π_t *is* ***finer*** *than* Π_s, *denoted by* $\Pi_s<\Pi_t$, *i.e. every atom of* Π_s *is a union of atoms of* Π_t.

The family of partitions (Π_t) *is called the* ***information flow*** *of the history.*

The terminology is borrowed from [Kahn 1974], and is in accordance with the usual notion of history, as the following examples will show.

2.1.1.1 Notations.

1/ Given a function X defined on Ω, we shall write for short

$$X\in\Pi_t \tag{2-1}$$

to indicate that X is constant on the atoms of the partition Π_t; in the same way, if A is a subset of Ω, we shall write

$$A\in\Pi_t \tag{2-2}$$

to indicate that A is a union of atoms of the partition Π_t; these notations will be extended to the

other partitions we shall encounter in the sequel.
2/ Given a logical proposition $P(\omega)$ depending on ω, we shall write for short

$$P \text{ instead of } \{\omega : P(\omega)\} \tag{2-3}$$

For example, $\{X > \lambda\}$ denotes the set $\{\omega : X(\omega) > \lambda\}$.

2.1.1.2 Two simple examples.

History associated to the instruction (o).

Consider the instruction

y = f(x(1),...,x(n))

We shall write X for short to denote the n-uple (x(1),...,x(n)). Introduce the function set

$$\Omega = \{\mathbf{N}_+ \to \Xi\} \tag{2-4}$$

where Ξ denotes the set in which the signal X takes its values, and denote by ω the elements of Ω: every ω represents thus a possible trajectory $\xi_1, \xi_2, \ldots$ of the input X. Then, endow Ω with the following equivalence relation

$$\{\omega \approx_t \omega'\} \Leftrightarrow \{\omega(s) = \omega(s') \ \forall s \leqq t\} \tag{2-5}$$

The partition Π_t is defined as follows: ω and ω' belong to the same atom of Π_t iff $\omega \approx_t \omega'$. By convention, $\Pi_0 = \{\varnothing, \Omega\}$ represents the information available before the time starts (constants,...), which do not depend on the values of the inputs. Then $\{\Omega, (\Pi_t)_{t \in \mathbf{N}}\}$ is the *canonical history associated to the instruction (o)*. The partition Π_t is interpreted as *the information available at time t*, i.e. any data available at time t in the system is nothing but a function defined on the set Ω, which is constant on the atoms of the partition Π_t, in other words a function of the initial segment $X_{[1,t]}$.

HIstory associated to the instruction (i).

The canonical history associated to the instruction (i) (the «register») is built in the same way, but, we should take here $\Omega = \{\mathbf{N} \to \Xi\}$, where Ξ is the space of the values of x; here, every ω represents $u, \xi_1, \xi_2, \ldots$, where $\xi_1, \xi_2, \ldots$ represents a possible trajectory of the input x, while u is a possible initial condition. In this case, Π_0 is the partition associated to the equivalence relation $\omega \approx_0 \omega' \Leftrightarrow u = u'$.

The instructions (ii) to (iv) will require the notions of clock and signal we shall introduce now.

2.1.2 Clocks.

Our notion of clock is apparently different from, but in fact closely related to the notion of event used in [Caspi & Halbwachs 1986]. Our main contribution is that we consider here clocks as functions, which map the input values ω into the set of the events (in the sense of Caspi & Halbwachs), and furthermore satisfy suitable causality constraints.

2.1.2.1 Definition of clocks.

A clock relates any sequence of events to a given time reference, by specifying at which date the s−th event occurs.

Definition 2: *Given a history* $\{\Omega, (\Pi_t)\}$, *a* $\{\Omega, (\Pi_t)\}$-***clock*** H *(or, simply a clock, when there is no ambiguity) is a time-indexed family of functions*

$$H_t: \ \Omega \rightarrow \Theta \tag{2-6}$$

where Θ *is a denumerable, totally ordered set containing* **N**, *and isomorphic to* **N**, *which satisfies the following properties* $\forall\omega$

$$\begin{aligned} &H_0(\omega) = 0, \quad H_\infty(\omega) = \infty && (i)\\ s < t \quad &\text{and} \quad H_s(\omega) < \infty \Rightarrow H_s(\omega) < H_t(\omega) && (ii)\\ s \leqq H_t(\omega) < s+1 \quad &\text{and} \quad \omega' \approx_s \omega \Rightarrow H_t(\omega') = H_t(\omega) && (iii) \end{aligned} \tag{2-7}$$

Recall that $H_s(\omega) = \infty$ means that the s−th event of the clock H never happens. The last condition is a causality condition: it expresses the fact that, to know if the t-th occurrence of an event $H(\omega)$ takes time strictly before $s+1$, it is sufficient to observe the initial segment up to time s. Finally, since our aim is to avoid, or at least detect nondeterminism, any multiple occurrence of an event must be time ordered; hence, the use of oversets of **N** is necessary, since we want to handle multiple events. The relevance of the time correctness constraint (2-7) will be subsequently illustrated in the forthcoming examples and counterexamples.

2.1.2.2 Counters.

Counters count the number of events that occur before a given instant of some time reference. Counters are extensively used in [Caspi & Halbwachs 1986] to monitor events in real time systems. Counters are associated to clocks as follows. Given a $\{\Omega, (\Pi_t)\}$-clock H, the following formula, where # stands for short for «cardinal of»,

$$\mu_t^H(\omega) = \#\{s \in \mathbf{N} : H_s(\omega) < t+1\} \tag{2-8}$$

defines the *counter associated to H.* Note that it is generally not possible to recover H from its counter unless $\Theta = \mathbf{N}$, since counters do not provide any information about the ordering of the events which occur between t and $t+1$; this would cause difficulties when time changes are of interest, as we shall see later. Thanks to (2-7−iii), counters enjoy the following property:

Lemma 1: we have

$$\{\omega' \approx_t \omega\} \Rightarrow \{\mu_t^H(\omega') = \mu_t^H(\omega)\}$$

The proof is easy, and is left to the reader. This lemma expresses that the counter associated to a clock is a causal signal in the sense of the definition 5 below.

2.1.2.3 Traces.

The notion of trace we shall now introduce is in fact the proper framework to handle simultaneously both dual aspects of time, namely clock/counter.

Definition 3: *the trace associated to a clock H is the subset $\overline{H}$ of $\Omega\times\Theta$ defined by*

$$(\omega,\theta)\in\overline{H} \Leftrightarrow \exists t: H_t(\omega)=\theta \tag{2-9}$$

It will be convenient to introduce the following notations:

$$\begin{aligned}\overline{H}(\omega,.) &= \{\theta\in\Theta: (\omega,\theta)\in\overline{H}\}\\ \overline{H}(.,\theta) &= \{\omega\in\Omega: (\omega,\theta)\in\overline{H}\}\end{aligned} \tag{2-10}$$

The interest of this notion is that the clock and the counter can be easily recovered from it, as we shall explain now.

$$\begin{aligned}\mu_t^H(\omega) &= \#\{\overline{H}(\omega,.)\cap[0,t+1)\}\\ H_t(\omega) &= \min\left\{\theta:\ \#\{\overline{H}(\omega,.)\cap[0,\theta]\}=t\right\}\end{aligned} \tag{2-11}$$

Moreover, we have the following result:

Lemma 2: Given a subset $\overline{H}$ of $\Omega\times\Theta$, the formulas (2-11) define a clock iff the following property holds

$$\forall\theta<s+1:\ \overline{H}(.,\theta)\in\Pi_s \tag{2-12}$$

Here $\overline{H}(.,\theta)\in\Pi_s$ means as usually that the referred subset of Ω is a union of atoms of Π_s. The proof is immediate.

2.1.2.4 Examples and counterexamples.

Example 1: any stricly increasing function $H: \mathbf{N}\to\mathbf{N}$ which does not depend on the input stimuli ω is a clock, since the causality constraint (2-7-iii) is trivially satisfied. The events generated by such clocks are known prior the observation of the input stimuli, i.e. before the running of the program.

Example 2: take $\Theta=\mathbf{N}^2$ endowed with the lexicographic order, defined by

$$(m,m')<(n,n') \Leftrightarrow [m<n] \quad \text{or} \quad [m=n \quad \text{and} \quad m'<n']$$

Fix p, and set

$$H_t = (s,k) \text{ if } t = sp + k \text{ with } 0 \leqq k < p$$

This is the simplest case of *oversampling*, where the new sampling rate does not depend on the input stimuli. This procedure will be properly generalized later.

Example 3: consider a causal signal X, which means in our framework that

$$\omega' \approx_t \omega \Rightarrow X_t(\omega) = X_t(\omega') \tag{2-13}$$

and let A be a subset of the domain in which X takes its values. Define

$$\begin{aligned} H_0(\omega) &= 0 \\ H_1(\omega) &= \min\{s > 0 : X_s(\omega) \in A\} \\ H_{t+1}(\omega) &= \min\{s > H_t(\omega) : X_s(\omega) \in A\} \end{aligned} \tag{2-14}$$

where, by convention, $\min(\varnothing) = +\infty$. Then, H is a clock, for it satisfies the constraint (2-7-iii) thanks to (2-13).

This example can also be written in the mini language SL_2; we assume for example that X takes real values, and that $A =]\lambda, \infty[$. The corresponding instruction is

H = *true* **when** X>λ

The set of the occurrences of the signal H defines exactly the trace of the clock H introduced in (2-14).

Example 4: here follow two counterexamples. First, modify (2-14) as follows:

$$\begin{aligned} H_0(\omega) &= 0 \\ H_1(\omega) &= \min\{s > 0 : X_{s+u}(\omega) \in A\} \\ H_{t+1}(\omega) &= \min\{s > H_{t+u}(\omega) : X_s(\omega) \in A\} \end{aligned}$$

where $u > 0$. Then, H is clearly a $\{\Omega, (\Pi_{t+u})\}$-clock, but not a $\{\Omega, (\Pi_t)\}$-clock, since the time-correctness constraint is violated. More subtle is the following example: let $k_1, k_2, \ldots$ be an increasing sequence of integers; define

$$H_t(\omega) = \max\{s < k_t : X_s(\omega) \in A\}$$

with the convention $\max(\varnothing) = 0$. Then, H is not a clock, since the causality constraint is violated. Such counterexamples cannot be described using the instructions of SL_i, since they require, to some extend, to be able to talk about the future, which is not possible in SL_i, since \$ allows only to talk about the past, and other elementary instructions refer to the present.

2.1.2.5 History associated to a clock.

This notion will be a first step towards the technique of time change. The idea supporting time changes is the following: suppose you have a clock, and you are interested in an infinite ordered file of data which are available at each occurrence of this clock. Then, you would probably like to forget the original time reference, and prefer to work with the above mentioned clock *as it were the time reference.* Our aim is to justify such a procedure. A first step in this direction is the following definition.

Definition 4: *Let H be a $\{\Omega,(\Pi_t)\}$-clock. Define*

$$\omega \approx_{H_t} \omega' \Leftrightarrow \exists s:\ s \leqq H_t(\omega) < s+1 \text{ and } \omega \approx_s \omega' \qquad (2\text{-}15)$$

This is an equivalence relation; (2-15) defines (Π_{H_t}) as the ***history associated to the clock*** *H.*

The fact that (2-15) defines an equivalence relation is due to the property (2-7-iii) of time-correctness of the clock H. The corresponding information flow summarizes the flow of information corresponding to every new occurrence of H, a fundamental notion when the study of time changes is of interest. This notion is the good functional point of view to encode the notion of initial segment in the case of time changes.

2.1.3 Causal signals.

Consider an history $\{\Omega,(\Pi_t)\}$. For every $\omega\in\Omega$, we shall denote by

$$\omega/t \qquad (2\text{-}16)$$

the atom of Π_t which contains ω; this is nothing but the initial segment of ω up to time t in the case of the canonical process associated to an instruction of SL_i so that we shall extend the name of *initial segments* to refer to such objects. The set of the initial segments of Ω is endowed with the following partial order:

$$\omega/t_1 \leqq \omega/t_2 \Leftrightarrow t_1 \leqq t_2 \text{ and } \omega/t_2 \subset \omega/t_1 \qquad (2\text{-}17)$$

Then, every chain $\omega/t_1 \leqq \omega/t_2 \leqq \ldots$ has a unique maximal element, which is nothing but the atom $\cap_t \omega/t$. This upper limit defines a unique point of Ω when the atoms of $\cup_t \Pi_t$ are the points of Ω, which is for example the case for the canonical process of an instruction. Then, it is reasonable to define a causal signal X as a time indexed set of functions of ω such that $X_t(\omega)$ depends only on the initial segment ω/t. The following definition generalizes this idea.

Definition 5: *Let H be a $\{\Omega,(\Pi_t)\}$-clock. By a $\{\Omega,(\Pi_t),H\}$-signal* (for short instead of «causal signal»), *we have in mind a time-indexed family $X=(X_t)_{t\in\mathbf{N}_+}$ of functions*

$$X_t : \Omega \rightarrow \Xi$$

(where Ξ is a set defining the type of the signal), *satisfying the following* ***time-correctness*** *condition:*

$$\omega' \approx_{H_t} \omega \quad \Rightarrow \quad X_t(\omega') = X_t(\omega) \tag{2-18}$$

The definition 4 expresses the fact that $X_t(\omega)$ depends only on the initial segment $\omega/H_t(\omega)$; this initial segment is well defined since H is a clock; roughly speaking, the value X_t has to be considered as available at time H_t; in this case, $H_t = \infty$ means that X_t is never observed.

An example using the mini language SL_2.

The SL_2 instruction

y = x **when** x>0

produces the signal $y_t(\omega) = x_{H_t}(\omega)$, where ω denotes a trajectory of the input x, and H_t denotes the clock ;ofthe occcurrences of the event x>0.

2.2 The clock algebra.

The aim of this section is to study the algebra of $\{\Omega, (\Pi_t)\}$-clocks, we shall simply refer to as «clocks», since no confusion can occur throughout this chapter. We shall introduce two primitive operations on this set: the *filtering*, and the *multiplexing*. And we shall prove that these primitives allow us to build in finitely many steps any (reasonable) clock.

2.2.1 A partial order on the set of the clocks.

It is defined as follows. Given two clocks H and K, referring to (2-10), we define

$$K \subset H \Leftrightarrow \forall\omega : \overline{K}(\omega, .) \subset \overline{H}(\omega, .) \tag{2-19}$$

In other words, $K \subset H$ means that the set of occurrences of K is included in the set of the occurrences of K *whatever the input* ω *is*, i.e. $\subset$ is a partial order on functions. This order is different from the order introduced in [Caspi & Halbwachs 1986], the aim of which is the study of precedence of events. Since the union $\Theta \cup \Theta'$ of two totally ordered sets in which clocks take values is not necessarily totally ordered, *it is not always true that two arbitrary clocks K and H do possess a common upper bound* corresponding to the order (2-19). The operations of supremum (when it does exist) and infimum will be respectively denoted by

$$K \vee H, \quad K \wedge H \tag{2-20}$$

2.2.2 The filtering.

Every signal of boolean type will be said to be a *test*. In this paragraph, we shall sometimes identify the boolean values *true* and *false* respectively with the integer values 1 and 0. The following lemma holds:

Lemma 3: Let H be a clock, and T be a $\{\Omega, (\Pi_t), H\}$-test. Then, the following formula

$$K_t(\omega) = \min\{H_s(\omega) : \sum_{u=1}^{s} T_u(\omega) \geqq t\} \tag{2-21}$$

defines a new clock, denoted by

$$K = H \downarrow T \tag{2-22}$$

and referred to as the clock obtained by filtering H by T.

Proof: we shall make use of the notations (2-1,2-3). We have

$$\begin{aligned} &\{v \leqq K_t < v+1\} \\ &= \bigcup_s \left(\{ \sum_{u=1}^{s-1} T_u < t \leqq \sum_{u=1}^{s} T_u \} \cap \{v \leqq H_s < v+1\} \right) \\ &:= \bigcup_s \left(A_s \cap B_s \right) \end{aligned}$$

Since H is a clock, we have on one hand $\forall s: B_s \in \Pi_v$. On the other hand, since T is a causal signal, $\forall s: A_s \in \Pi_{H_s}$ holds. But, since H is a clock, it is true that the restriction to the set B_s of the partitions Π_v and Π_{H_s} are identical; hence $A_s \cap B_s \in \Pi_v$, which finally implies that $\{v \leqq K_t < v+1\} \in \Pi_v$, i.e. K is a clock. □

The meaning of the filtering is as follows: *$H \downarrow T$ extracts from H the instants H_s where T_s is true.* Conversely, the following result holds, where we have used the notation (2-10):

Lemma 4: If $K \subset H$ in the sense of (2-19), then $K = H \downarrow T$ holds, where the $\{\Omega, (\Pi_t), H\}$-test T is given by

$$T_s(\omega) = \begin{cases} \textit{true} & \text{if } H_s(\omega) \in \overline{K}(\omega, .) \\ \textit{false} & \text{otherwise} \end{cases} \tag{2-23}$$

The proof is elementary, and is left to the reader.

Example: The filtering is the basic operation to build the set of all the clocks which are dominated by a single one. This operation is available in all the above mentioned languages ESTEREL, LUSTRE, and SIGNAL. Taking the example of the mini language SL_2 if x and b denote two signals subject to the same clock H the instruction (iv) "y = x **when** b" generates exactly the clock $H \downarrow b$.

2.2.3 The multiplexing.

When the oversampling of data rates is of interest (and this is actually often the case), no tool is usually provided in real time oriented languages to handle properly the corresponding time index. Our purpose is now to investigate theoretically the difficulties behind the notion of oversampling. The corresponding operation on clocks will be called ***multiplexing***, for short instead of time-multiplexing.

Let H be a clock taking values in an overset Θ of $\mathbf{N}$ (see the definition 2), and let C be a causal $\{\Omega, (\Pi_t), H\}$-signal of integer type, such that $C_t \geqq 0$ for t finite, and $C_\infty = 0$. Set

$$\Theta' = \Theta \times \mathbf{N} \tag{2-24}$$

endowed with the *lexicographic* order, defined by

$$(t,k) < (t',k') \quad \Leftrightarrow \quad t < t' \text{ or } (t = t' \text{ and } k < k') \tag{2-25}$$

where, by convention, $(\infty, k) = \infty \ \forall k$. Notice that Θ is naturally identified with the subset Θ x $\{0\}$ of Θ', we shall often use this natural embedding in the sequel.

Definition 6: *The multiplexing of the clock H by the* ***ceiling***[1] ***function*** *C is defined as follows:* $K_0(\omega) = 0$, *and, for* $t > 0$

$$K_t(\omega) = \begin{cases} (H_s(\omega), u) \text{ if } 0 \leqq u \leqq C_s(\omega) \text{ and } t = 1 + u + \Sigma_1^{s-1}(C_v(\omega) + 1) \\ \infty \text{ otherwise} \end{cases} \tag{2-26}$$

and is denoted by

$$K = H \uparrow C \tag{2-27}$$

In (2-26), by convention, $\Sigma_1^0 = 0$ The figure 1 depicts this procedure: the $\uparrow$ arrows denote an increase by 1 of the second component of the clock $H \uparrow C$, whereas the $\downarrow$ replaces at the same time H_s by H_{s+1}, and $u = C_s$ by $u = 0$.

To justify the definition above, we have to prove the following theorem:

---

[1] this name, originated from the architecture theory, is justified by the figure 1, where the depicted object looks somewhat like Manhattan

Theorem 1: *$K = H \uparrow C$ is a $\{\Omega, (\Pi_t)\}$-clock.*

Proof: referring to (2–26), and using (2–25) and the notations (2–1,2–3), we have

$$\{\nu \leqq K_t < \nu + 1\} = \{\nu \leqq H_s < \nu + 1\} \tag{2-28}$$

Choose $\omega \approx_\nu \omega'$; then (2–28) and the fact that C is a signal of clock H implies that

$$\begin{aligned} H_s(\omega') &= H_s(\omega) \\ C_k(\omega') &= C_k(\omega) \quad \forall k \leqq s \end{aligned} \tag{2-29}$$

Finally, (2–26) and (2–29) imply the desired property, namely $K_t(\omega) = K_t(\omega')$, this finishes the proof.

The next theorem is the first fundamental result. It expresses the fact that allmost all $\{\Omega, (\Pi_t)\}$-clock can be obtained by applying finitely many times the operations ***filtering*** and ***multiplexing***.

Theorem 2: *Let H be a $\{\Omega, (\Pi_t)\}$-clock taking values in the set*

$$\Theta = \mathbf{N} \times \mathbf{N}^{k-1}, 0 < k < \infty \tag{2-30}$$

endowed with the lexicographic order. Then, H can be decomposed as follows

$$\begin{aligned} H(0) &= Id && (i) \\ H(1) &= H(0) \downarrow T(1) && (ii) \\ \text{for } m > 0, H(m+1) &= H(m) \uparrow C(m) \downarrow T(m) && (iii) \\ H &= H(k) && (iv) \end{aligned} \tag{2-31}$$

where T(m) (resp. C(m)) are suitable tests (resp. ceiling functions), and Id denotes the clock «identity», defined by $Id_t(\omega) = t$. Furthermore, among all the possible decompositions (2–31), there is a ***minimal*** *one, we shall denote by $H^*(0), \ldots, H^*(k)$, such that*

$$H^*(m) \subset H(m) \tag{2-32}$$

for any decomposition associated to H through (2–31).

COMMENT 1: Θ's of the form mentioned in the theorem are not the most general, so that the theorem does not cover all the possible clocks. However, $\mathbf{N}^{\mathbf{N}}$ endowed with the lexicographic order is nothing but the (non denumerable) set of the denumerable ordinals, which is of no practical interest for us.

Proof: Let us denote by $proj_m$ the projection map of Θ onto $\mathbf{N} \times \mathbf{N}^{m-1}$, where we mean here that we keep the m first coordinates of θ. Using the notations of (2-10), we set

$$\overline{H}(m) = proj_m(\overline{H}) \tag{2-33}$$

and define $H(m)$ using the formula (2-11). To show (2-31), it clearly suffices

1. to verify that $\overline{H}(m)$ satisfies the condition (2-12)
2. to prove that $H(m+1)$ and $H(m)$ are related through (2-31-iii).

The first assertion is proved by induction over m; decompose $\theta(m+1) = (\theta(m), u)$ and consider s such that $s \leqq \theta(m) < s+1$; the definition of the lexicographic order implies that $s \leqq \theta(m+1) < s+1$ also holds. The formula

$$\overline{H}(m) = proj_m[\overline{H}(m+1)] \tag{2-34}$$

implies in this case that

$$\begin{aligned}\overline{H}(m)(., \theta(m)) &= proj_m[\overline{H}(m+1)(., \theta(m+1))] \\ &= \bigcup_u \overline{H}(m+1)[., (\theta(m), u)] \in \Pi_s\end{aligned}$$

which proves the first assertion by induction. We shall now prove the second assertion. Take

$$C(m)_t(\omega) = \max\{u \geqq 0 : (\theta(m), u) \in \overline{H}(m+1)(\omega, .)\}$$

and apply lemma 3 to the clock $H(m)$ which is dominated by $H(m) \uparrow C(m)$ to get $T(m)$. This finishes the proof of the existence of the decomposition (2-31).
We shall now prove that (2-33) provides indeed the minimal decomposition $H^*(m)$. This is due to the fact that any decomposition (2-31) must satisfy the inclusion relationship $\overline{H}^*(m) \subset \overline{H}(m)$ since we must have ultimately the equality $\overline{H}^*(k) = \overline{H}(k)$. This finishes the proof of the theorem.

FUNDAMENTAL REMARKS.

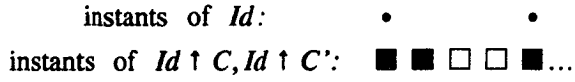
1/ Some existing real time synchronous languages explicitely assume that every clock is defined in terms of the *finest* one. The usual underlying argument is that, when a new faster clock has to be introduced in the program, then all the events have to be reexpressed with respect to this faster time reference. This argument is incorrect, as shown by theorems 1 and 2 above: *the master clock* (when it exists) *is not necessarily the fastest one, but is rather the clock from which every other one can be derived.* For example, if C is an input dependent ceiling function, the clock $Id \uparrow C$ (where Id denotes the identity clock) is faster than Id, but uses C to be built, so that $Id \uparrow C$ is derived from Id, while the converse is not possible.

2/ The combined use of the operations $\uparrow$ and $\vee$ can cause difficulties, as the

following example shows. Consider two different ceiling functions C and C'. Should we consider that the two clocks $Id \uparrow C$ and $Id \uparrow C'$ take their values

1. in the *same*,
2. or in *different*

copies of the set $\mathbf{N} \times \mathbf{N}$? In the case 1, $(Id \uparrow C) \vee (Id \uparrow C')$ does exist, while it does not in the case 2 since no total order is defined on $\Theta \cup \Theta'$. Although simpler, the first choice is not very convenient, for it would result in very strange situations: take $C \equiv 1$ and $C' \equiv 3$, which would intuitively correspond to multiplying the sampling rate by 2 and 4 respectively, but the instants of $Id \uparrow C$ would not be obtained by selecting every second instant of $Id \uparrow C'$, as the following picture shows (the instants of $Id \uparrow C$ and $Id \uparrow C'$ are superposed, and are depicted by □'s by ■'s respectively):



Finally, the most reasonable choice is 2, namely to always assume that *different ceiling functions create clocks taking values in different totally ordered sets.* This should be kept in mind in the sequel when referring to the multiplexing.

An illustration using SL_3.

The instruction (iii) (the multiplexer) is the exact translation of the above introduced operation on clocks. The theorem 2 expresses that every synchronous real time language should provide a construct which covers the multiplexing.

2.2.4 Synchronizable clocks.

***Definition** 7: Two clocks H and H' are said to be **synchronizable** if their associated counters μ^H and $\mu^{H'}$ are equal.*

The name of «synchronizable» is justified by the following result:

***Lemma** 5:* If H and H' are synchronizable, then their associated histories $\{\Omega, (\Pi_{H_t})\}$ and $\{\Omega, (\Pi_{H'_t})\}$ are identical.

In other words, synchronizable clocks define the same flow of information.
Proof: since H and H' are synchronizable, we have, for every t, ω, $s \leqq H_t(\omega) < s + 1$ iff $s \leqq H'_t(\omega) < s + 1$, which proves the result, thanks to (2-15).

This lemma indicates to some extend what are the degrees of freedom in matching signals subject to different clocks: it suffices these clocks are synchronizable.

2.3 History communication and time changes.

2.3.1 Examples.

Example 1: refering again to SL_2, consider the program

```
y = x when x>0 ||
z = u + y
```

If t denotes the time index of the input "x" of this program, the clock of "y" is H_t, where $\overline{H}(\omega, .)$ is the set of the occurrences of the event x>0. On the other hand, the second instruction z=u+y is such that the three involved signals have the same time index, say "s". But the communication operator || results in the substitution $s \leftarrow H_s$. This illustrates the fact that signal communication and time changes are tightly related notions.

Example 2: Consider again the program of the time multiplexing. The kind of communication which is here exhibited is no more a «cascade» as in the previous example (by «cascading» two instructions, we have in mind that some of the outputs of the first instruction are connected to corresponding inputs of the second one, but not the converse). Here, every instruction possesses inputs and outputs that are connected to other outputs and inputs. Consequently, it is no more obvious *which clock will be the master one,* and thus serves at a reference for the definition of time changes[2].

To define history communication, we shall proceed as follows. First, we define the *history juxtaposition,* which is the result of observing two histories which don't communicate at all. Then, a simple restriction operation will provides us with the abstract notion of *history communication.*

2.3.2 History juxtaposition.

The idea behind history juxtaposition is simple. An observer which monitors two histories $\Omega\Pi$ and $\Omega\Pi'$ that do not exchange any information will observe the interleaving of the events belonging to $\Omega\Pi$ or $\Omega\Pi'$. We shall now formulate this idea rigorously by proceeding into two steps.

2.3.2.1 STEP 1: inserting dummy events.

Consider a history $\Omega\Pi = \{\Omega, \Pi\}$. Introduce the following objects.

$$\perp = \{\mathbf{N} \rightarrow \mathbf{N}\} \qquad (2\text{-}35)$$

Elements of $\perp$ are denoted by γ. Notice that γ is allowed to take the value ∞. Introduce

$$\Omega^{\perp} = \Omega \times \perp \qquad (2\text{-}36)$$

---

[2] although we expect here that this master clock should be the clock of the ceiling function C, since our purpose was to synthetize the multiplexing

COMMENT 2: The intepretation of $\Omega^{\perp}$ is as follows: elements of this set are of the form (ω, γ), where γ is a given sequence of nonnegative integers; $\gamma(t)$ specifies how many «dummy» events (i.e. events to which $\Omega\Pi$ do not participate) will be inserted between the instants t and $t+1$ of the history $\Omega\Pi$. Notice that *inserting* ∞ *dummy events means that* $\Omega\Pi$ *is waiting for ever for its next event, i.e.* $\Omega\Pi$ *starvates.*

Then, $\Omega^{\perp}$ will be endowed with the following information flow:

$$(\omega, \gamma) \approx_t (\omega', \gamma') \tag{2-37}$$

$$\Leftrightarrow$$

$$u_0 = \max\{u: \sum_{v=0}^{u} (1 + \gamma(v)) \leq t\} \Rightarrow \begin{cases} \forall u \leq u_0 : \gamma'(u) = \gamma(u) \\ \omega \approx_{u_0} \omega' \end{cases}$$

The figure 2 below depicts two trajectories of $\Omega^{\perp}$ that are equivalent in the above introduced sense.

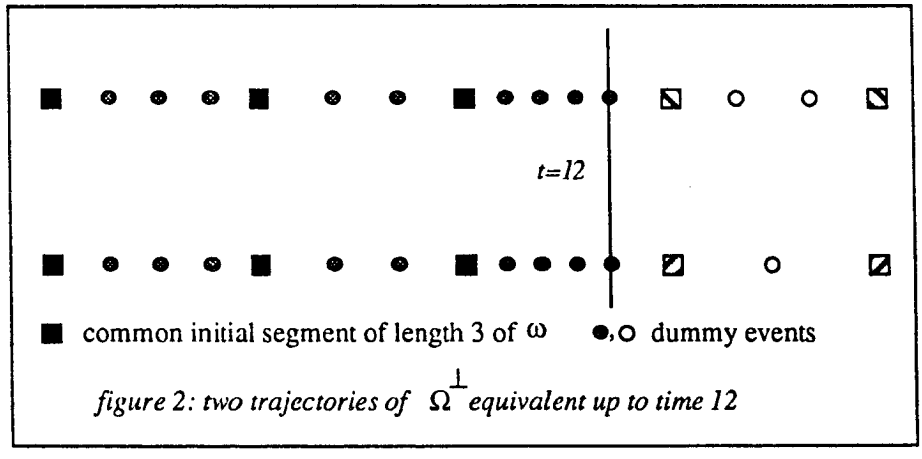


figure 2: two trajectories of $\Omega^{\perp}$ *equivalent up to time 12*

The corresponding information flow will be denoted by $(\Pi_t^{\perp})$.

REMARK: Notice that, although we have inserted dummy events between already existing ones, *it is not true that the new clock has been obtained using the multiplexing,* since the causality condition which is required for the ceiling function is not satisfied; roughly speaking, observing $\Omega\Pi$ up to the t-th event of this history does not provide any information about how many dummy events will be inserted after this t-th event. In fact, this property is at the origin of the possibility to introduce non-determinism, as we shall see later.

2.3.2.2 STEP 2: defining history juxtaposition.

Definition 8: Given two histories $\Omega\Pi = \{\Omega, \Pi\}$ *and* $\Omega\Pi' = \{\Omega', \Pi'\}$, *their* ***juxtaposition*** *is denoted by*

$$W\Gamma = \Omega\Pi \;\&\; \Omega\Pi' \tag{2-38}$$

and is defined by

$$\begin{gathered} W\Gamma = \{W, \Gamma\}, \quad \text{where} \\ W = \{((\omega,\gamma),(\omega',\gamma')) \in \Omega^{\perp}\times\Omega'^{\perp} :\ \gamma(t).\gamma'(t) = 0\ \forall t\} \\ \Gamma_t = \{\Pi_t^{\perp}\times\Pi_t'^{\perp}\}_{|W} \end{gathered} \tag{2-39}$$

Property: *The operation & is commutative and associative.*

Here, the product of partitions is defined in an obvious way,

$$\{\bullet\}_{|W}$$

denotes the restriction of $\{\bullet\}$ to W, and $\gamma.\gamma'$ is the standard product of integers. The definition of W simply means that we keep in W the subset of $\Omega^{\perp}\times\Omega'^{\perp}$ composed by the trajectories all the events of which belong either to the history $\Omega\Pi$ or to the history $\Omega\Pi'$. The history Γ corresponds to the flow of information which is received by a simultaneous observer of $\Omega\Pi$ and $\Omega\Pi'$.

The two following clocks of $W\Gamma$ will play a fundamental role in the sequel. Introduce first the clock H^{Ω} as being the clock associated to the trace of the events in which $\Omega\Pi$ is involved (see the ■ and □ in the figure 2):

$$\begin{gathered} \overline{H}^{\Omega} = \{(w,t) \in W\times\mathbf{N} :\ \exists u,\ \sum_{\nu=0}^{u}(1+\gamma(\nu)) = t\} \\ \text{where}\quad w = ((\omega,\gamma),(\omega',\gamma')) \end{gathered} \tag{2-40}$$

The fact that H^{Ω} is a clock is easy to verify, using lemma 2 and (2-37). The clock $H^{\Omega'}$ of the events belonging to $\Omega\Pi'$ is defined in the same way.

2.3.3 History communication.

2.3.3.1 Definition.

Definition 9: *A **communication** of* $\Omega\Pi$ *and* $\Omega\Pi'$ *is defined through the specification of a subset of the trajectory space W of the juxtaposition* $\Omega\Pi \,\&\, \Omega\Pi'$. *The history* Γ *is then restricted accordingly. The set of these history communications will be denoted by*

$$\Omega\Pi \,\|\, \Omega\Pi' \tag{2–41}$$

For example, $\Omega\Pi \,\&\, \Omega\Pi'$ is a particular «communication» of $\Omega\Pi$ and $\Omega\Pi'$, namely the weakest one, in which $\Omega\Pi$ and $\Omega\Pi'$ do not communicate at all.

2.3.3.2 Examples again.

Cascades.

Consider again $\Omega\Pi$ and $\Omega\Pi'$, and assume a map

$$\Phi : \Omega \to \Omega'$$

is given which satisfies the following causality condition

$$\omega \approx_t \omega' \;\Rightarrow\; \Phi(\omega) \approx_t \Phi(\omega')$$

Then, set

$$W = \text{graph of } \Phi$$

The so-obtained communication of $\Omega\Pi$ and $\Omega\Pi'$ is in fact isomorphic to $\Omega\Pi$ (by «isomorphism», we mean a bijection which preserves histories) via the map

$$\omega \mapsto (\omega, \Phi(\omega))$$

This example is exactly the abstract translation in our framework of the intuitive notion of cascade of two histories, as the investigation of the following SL_0-example shows:

$$\begin{array}{l} x = a + b \\ \| \\ y = x + a \end{array}$$

In this example, we have two single clocked histories corresponding to the two instructions: denotes them by $\Omega\Pi$ and $\Omega\Pi'$; but, when ω is given, we know the trajectory of the signals a and b, hence of the output x, so that we know also the input trajectories of $\Omega\Pi'$, i.e. we know ω'.

This example could have been generalized to include the following SL_2-example, where we assume that the input signals a and b possess the same clock:

```
x = a when b
||
y = x__0 → $x
```

The history $\Omega\Pi = \{\Omega, \Pi\}$ corresponding to the first instruction is identical to the history corresponding to any function with a and b as inputs, since these signals are assumed to be simultaneous. Consider the map Φ which associates to a given input trajectory of $\Omega\Pi$ (observation of a and b) the corresponding trajectory of x. Since x is the input of the second instruction, a trajectory of x defines a point ω' of the corresponding history $\Omega\Pi'$. This map Φ satisfies the condition

$$\omega \approx_t \omega' \Rightarrow \Phi(\omega) \approx_{H_t} \Phi(\omega')$$

where H denotes the clock of the signal x (i.e. $H = Id \downarrow b$). Here, to specify the communication in $\Omega\Pi \,||\, \Omega\Pi'$, we take, according to the definition 9,

$$W = (\omega, \Phi(\omega))$$

so that $\Omega\Pi \,||\, \Omega\Pi'$ is again isomorphic to $\Omega\Pi$ via the map Φ. Notice that, according to (2–40), we have obviously $H^{\Omega} = Id \downarrow b$.

Closing circuits.

The next example is less trivial. Consider the following SL_1-program:

```
x = past__x + u
||
past__x = x__0 → $x
```

Here, the new difficulty lies in the fact that both signals x and past_x are at the same time input and output of both instructions: they define a communication between the corresponding histories $\Omega\Pi$ and $\Omega\Pi'$ which is no more a cascade. How to proceed? A trajectory ω of $\Omega\Pi$ is of the form

$$\omega = (p\_\_\xi, u) \tag{2–42}$$

where $p\_\_\xi$ and u denotes respectively two possible trajectories of the inputs past_x and u; on the other hand, a trajectory of $\Omega\Pi'$ is of the form

$$\omega' = \xi_0 \rightarrow \xi \tag{2–43}$$

where this notation means that the initial value ξ_0 is followed by a possible trajectory ξ of the input x. Referring again to the definition 9, to build the desired member of $\Omega\Pi \,||\, \Omega\Pi'$, it suffices to specify the subspace W; here, we take first the subset of the trajectory space of the juxtaposition $\Omega\Pi \& \Omega\Pi'$ of the trajectories w such that $\gamma \equiv 0$ and $\gamma' \equiv 0$, i.e. we do not need any dummy event (both histories are single-clocked); the corresponding trajectory space is obviously isomorphic to $\Omega \times \Omega'$. Second, this space is further reduced by keeping only the pairs (ω, ω') such that (see (2–42,2–43)

$$\forall t > 0: \begin{cases} \xi(t) = p\_\xi(t) + u(t) \\ p\_\xi(t) = \xi(t-1) \text{ if } t > 1, \ = \xi_0 \text{ if } t = 1 \end{cases}$$

It is clear that the resulting history is «nearly» isomorphic to the space of the trajectories of the input u (here, «nearly» means: up to the initial condition ξ_0) .

2.3.4 Embeddings.

The aim of this section is to answer the following question: referring to the notations of the preceding section, *is there any natural way to embed the clock (resp. signal) algebra of* $\Omega\Pi$ *into the corresponding algebras of* $W\Gamma$*?* A satisfactory answer to this question is a key pass to time changes.

Theorem 3:

(i) Let H *be a* $\{\Omega, \Pi\}$*-clock with values in the ordered set* Θ*. Denote by*

$$H = ((Id \downarrow T(1))....) \uparrow C(k) \downarrow T(k) \tag{2-44}$$

its minimal decomposition according to (2-30). Then, the formula

$$H^{\perp} = ((H^{\Omega} \downarrow T^{\perp}(1))....) \uparrow C^{\perp}(k) \downarrow T^{\perp}(k) \tag{2-45}$$

(where the operation $.^{\perp}$ on signals is defined in (ii) below) *defines a* $\{W, \Gamma\}$*-clock we shall call the* ***embedding of H into*** $W\Gamma$.

(ii) Let X *be a* $\{\Omega, \Pi, H\}$*-signal. The formula*

$$X_t^{\perp}(\omega, \gamma) = X_t(\omega) \tag{2-46}$$

defines a $\{W, \Gamma, H^{\perp}\}$*-signal we shall call the* ***embedding of X into*** $W\Gamma$.

Proof: By coupled induction over the lenght of the decomposition of the clock H; for H as in (2-44), this length is defined as $2k+1$, i.e. the total number of tests and ceiling functions required to build H in its minimal decomposition.

STEP 1: initial stage of the induction. Let us begin with the proof of (ii) for a signal with clock Identity. The proof is immediate, and the clock of the embedded signal is H^{Ω}, i.e. the clock of the events belonging to $\Omega\Pi$. On the other hand, assertion (i) for $H = Id$ is already known, since $H^{\perp} = H^{\Omega}$ in this case. In other words, (i) and (ii) are proved for a clock H with a decomposition of length 0.

STEP 2: induction. Assume (i) and (ii) have been proved for a clock H with decomposition of length $m \geqq 0$. Then, (i) results first for a clock H with decomposition of length $m+1$, and second, (ii) follows immediately in the same case.□

Notation: when there is no ambiguity, we shall no more distinguish between signals or clocks and their respective embeddings, hence we shall drop the superscript $\bullet^{\perp}$ when referring to these embeddings.

2.3.5 Determinism.

2.3.5.1 Example pursued.

Consider again a subset of the multiplexing program:

(i1) u = zu + 1
‖
(i2) zu = raz **default** past_u
‖
(i3) past_u = u_0 → $u

As one can guess (this will be formally verified later), the information flow Γ of the history associated to this program corresponds to *all the events to which anyone of the signals* u, zu, raz, *participates.* But it turns out that this program possesses only the signal raz as input, which generates an information flow which is *strictly smaller* than Γ in an intuitive sense (which will be formalized later). This discrepancy between information flows is reflected in the fact that *the input-output map* raz → (u,zu) *is nondeterministic:* this happens since the amount of events between two successive occurrences of raz is not specified by the observation of raz itself.

This is exactly the way the paradox about «how synchronous deterministic process communication can generate nondeterminism» was introduced in [Brock & Ackerman 1981]. We shall now further investigate this point.

2.3.5.2 Temporal nondeterminism, and observer's viewpoint.

Definition 10: *Consider a history* $\{\Omega, \Pi\}$, *and denote by* clock($\{\Omega, \Pi\}$) *the algebra of its clocks.*

(i) An information flow (Γ_t) *on* Ω *is said to be an* ***observer*** *of* $\{\Omega, \Pi\}$ *if*

$$\exists H \in \text{clock}(\{\Omega, \Pi\}): \quad \Gamma_t \subset \Pi_{H_t} \tag{2-47}$$

(ii) Given an observer Γ, *if the following condition holds*

$$\left\{\forall \omega, \forall s: H_t(\omega) \leqq s < H_{t+1}(\omega)\right\} \Rightarrow \left\{\omega' \approx_{\Gamma_t} \omega \Rightarrow \omega' \approx_{\Pi_s} \omega\right\} \tag{2-48}$$

then, the history $\{\Omega, \Pi\}$ *is said to be* ***deterministic with respect to the observer*** Γ, *while it is said to be* ***nondeterministic*** *otherwise.*

In (2-48), the relation $\approx_{\Gamma_t}$ means that the considered trajectories belong to the same atom of the partition Γ_t.

COMMENT 3: The condition (i) expresses that what an observer knows about the considered history is contained in the set of the flashes on this history at the events of a given clock. The condition (ii) expresses that the information flows Γ and Π give the same information about the considered history. A discrepancy between both information flows mean that there exists clocks or signals on this history which cannot be separated by the considered observer: this is the proper notion of nondeterminism. As the example above showed, this situation can occur even in simple cases. The interest of this rather abstract notion of determinism is that it will lead very easily to effective criteria to recognize it.

2.4 History congruence.

As we shall see later, it is interesting to identify histories which cannnot be distinguished by any observer; such histories will be said to be congruent. Before defining congruence, we shall need the stronger notion of isomorphism.

2.4.1 Isomorphisms.

Definition 11: *Two histories* $\{\Omega, \Pi\}$ *and* $\{\Omega', \Pi'\}$ *are said to be* ***isomorphic*** *if there exists a bijection* $\Phi : \Omega \to \Omega'$ *such that* $\Phi(\Pi_t) = \Pi'_t \; \forall t$.

2.4.2 Congruence.

We shall need the following notation:

$$\{\Omega, \Pi\} / \Pi_\infty \tag{2-49}$$

will denote the history obtained by taking the quotient set Ω/Π_∞ as trajectory space, and modifying the information flow Π accordingly.

Definition 12: *Two histories* $\{\Omega, \Pi\}$ *and* $\{\Omega', \Pi'\}$ *are said to be* ***congruent****, denoted by*

$$\{\Omega, \Pi\} \sim \{\Omega', \Pi'\} \tag{2-50}$$

if the quotient histories $\{\Omega, \Pi\} / \Pi_\infty$ *and* $\{\Omega', \Pi'\} / \Pi'_\infty$ *are isomorphic.*

The name of «congruence» is justified by the fact that the operations of history juxtaposition and history communication are compatible with the relation $\sim$.

2.5 Summary of the chapter, and discussion.

We have first introduced the notion of history to investigate what can be built when the set of the available stimuli is fixed. We have analysed the algebras of clocks and of signals. Basic operations of filtering and multiplexing were introduced, and it has been proved that these operations are the convenient primitives to construct all the clocks of a given history.

Then, we have introduced the notion of history communication, and derived associated embeddings for the clocks and signals of the original histories. Finally, we have introduced an elegant definition of determinism/nondeterminism.

But our definition of history communication is by no means effective, since we never said how the subset W in the definition 9 should be constructed. There are several possible ways to specify systematic rules to construct W: the purpose of the next chapter is to present one of these, which is closely related to the family of languages SL_i.

Chapter Three
THE MODEL OF PROCESSES.

The model of processes will be now introduced without illustrations via examples. The reason is that, first we hope the reader to be now more familiar with the technique, and second we shall use this model of processes to build a denotational model of the language SL_4.

3.1 Processes.

3.1.1 Definition of processes.

Definition 13: *A process is a triple*

$$P = \{\Omega, \Pi, \mathbf{A}\} \tag{3-1}$$

here, $\{\Omega, \Pi\}$ *is an history, and* $\mathbf{A}$ *is a finite collection of* ***ports****, i.e. triples of the form*

$$[a, X, H] \tag{3-2}$$

where "a" is a name, H a clock, and X a signal with clock H. We shall say that the ***port*** *named "a" carries the signal X with clock H.*

We have obviously in mind that process communication (we shall define next) can occur through ports only. Referring to this terminology, the names occurring in the instructions of the mini-languages SL_i are port names, and these instructions specify relationships between the signals carried by these ports, and between their respective clocks.

3.1.2 History generated by a family of ports.

Consider a process $P = \{\Omega, \Pi, \mathbf{A}\}$, and a subset $\mathbf{B}$ of $\mathbf{A}$. We shall denote by

$$o\,(\mathbf{B}) \tag{3-3}$$

the smallest information flow with respect to which all elements of $\mathbf{B}$ be respectively signals and clocks. Such a minimal information flow does exist, and is the infimum of the set of the sub-information flows of Π such that all elements of $\mathbf{B}$ satisfy the usual causality properties.

3.2 Process communication.

Definition 14: *Assume two processes* $P = \{\Omega, \Pi, \mathbf{A}\}$ *and* $P' = \{\Omega', \Pi', \mathbf{A}'\}$ *are given; denote by* **B** *and* **B'** *respectively the subsets of the members of* **A** *and* **A'** *which possess the same name in both processes. Then the* ***communication*** *of P and P' is denoted by*

$$Q = P \parallel P' \tag{3-4}$$

and is defined as follows, referring to the notations of definition 9 and theorem 3:

$$Q = \{W, \Gamma, \mathbf{C}\}, \tag{3-5}$$

where W is the subset of the trajectories of $\{\Omega, \Pi\} \& \{\Omega', \Pi'\}$ *such that*

$$\forall [b, X, H] \in \mathbf{B},\ \forall [b, X', H'] \in \mathbf{B}':\ \begin{cases} X'_t(w) = X_t(w) \\ H'_t(w) = H_t(w) \end{cases} \tag{3-6}$$

and

$$\mathbf{C} = \mathbf{A} \cup (\mathbf{A}' - \mathbf{B}')$$

Property: *Process communication is commutative and associative.*

Here, we have used the sketchy notations to refer to embedded clocks and signals, i.e. we have dropped the superscripts $\bullet^{\perp}$. Finally, we identify the connected ports in the resulting process.

COMMENT 4: This definition expresses that processes do communicate via ports with the same name. This communication constraints the clocks of the corresponding carried signals to be identical, and requires these signals to be equal. Notice that W is never void, since it contains at least w's with infinitely many inserted dummy events.

COMMENT 5: Let us emphazise that communicating via ports with identical names is only a possible way among many others to specify process communication. For instance, in our case, process communication results in a set of constraints between the involved ports, i.e. our model is a *relational* one. It should have also been possible to distinguish «input» and «output» ports, and to allow only «input ↔ input» or «output → input» connections. Such a technique is required if the purpose is to build a functional model; notice that the languages ESTEREL, LUSTRE, SIGNAL, and the family SL_i are functional languages. The reason for which we preferred to deal with a relational model is to avoid to handle problems related to data and control dependencies, a task which would have made our model more complicated (see for example [Le Guernic & Benveniste 1986]). We preferred to concentrate on a fundamental study of synchronization mechanisms.

3.3 Process congruence.

Following [Milner 1982] and [Austry & Boudol 1983], we introduce a notion of equivalence with respect to a set of observable ports.

Definition 15:

(i): Two processes $P = \{\Omega, \Pi, \mathbf{A}\}$ *and* $P' = \{\Omega', \Pi', \mathbf{A}'\}$ *are said to be* ***isomorphic*** *if there exist bijections* $\Psi: \mathbf{A} \to \mathbf{A}'$ *and* $\Phi: \Omega \to \Omega'$ *such that*

$$\forall [a, X, H] \in \mathbf{A}: \begin{cases} \Phi(\Pi_t) = \Pi'_t \\ (\Psi(X))_t(\Phi(\omega)) = X_t(\omega) \\ (\Psi(H))_t(\Phi(\omega)) = H_t(\omega) \end{cases} \tag{3-7}$$

where $\Psi(X), \Psi(H)$ *denote respectively the signal and clock of* $\Psi(\mathbf{A})$.

(ii): Two processes $P = \{\Omega, \Pi, \mathbf{A}\}$ *and* $P' = \{\Omega', \Pi', \mathbf{A}'\}$ *are said to be* ***congruent,*** *denoted by* $P \sim P'$ *if the quotient processes* $P/\sigma(\mathbf{A})$ *and* $P'/\sigma(\mathbf{A}')$ *are isomorphic, where*

$$P/\sigma(A) := \{\Omega/\sigma(\mathbf{A})_\infty, \sigma(\mathbf{A}), \mathbf{A}\}$$

Notice that inserted dummy events, as introduced for the definition of history juxtaposition, are ignored by the observation criterion corresponding to our congruence ~; this compares with the non observation of delays in the weak bisimulation of SCCS, [Milner 1982]. The congruence is compatible with process communication, i.e.

$$P \sim P' \text{ and } Q \sim Q' \Rightarrow P \parallel Q \sim P' \parallel Q'$$

As a consequence, in the sequel we shall no more distinguish congruent processes. Notice that restricting the observer to a smaller one ($\mathbf{B} \subset \mathbf{A}$) would cause the loss of the compatibility property; this is a wellknown situation.

The next section is a fundamental one. It is shown how effective criteria can be derived to decide whether the constraints imposed by a process communication can result in a starvation of (part of) the processes, or in nondeterminism.

3.4 The clock calculus.

3.4.1 Process synchronization.

Definition 16: Assume two processes $P = \{\Omega, \Pi, \mathbf{A}\}$ *and* $P' = \{\Omega', \Pi', \mathbf{A}'\}$ *are given; denote by* **B** *and* **B'** *respectively the subsets of the members of* **A** *and* **A'** *which possess the same name in both processes. Then the* ***synchronization*** *of P and P' is denoted by*

$$R = P \parallel_{synchro} P' \tag{3-8}$$

and is defined as follows, referring to the notations of definition 9 and theorem 3:

$$R = \{V, \Gamma, \mathbf{C}\}, \quad \text{where} \tag{3-9}$$

V is the subset of the trajectories of $\{\Omega, \Pi\} \& \{\Omega', \Pi'\}$ *such that*

$$\forall [b, X, H] \in \mathbf{B},\ \forall [b, X', H'] \in \mathbf{B}':\ \begin{cases} X'_t(w) = X_t(w) \\ H'_t(w) = H_t(w) \end{cases} \tag{3-10}$$

if the signals X and X' are boolean, and

$$\forall [b, X, H] \in \mathbf{B},\ \forall [b, X', H'] \in \mathbf{B}':\ H'_t(w) = H_t(w) \tag{3-11}$$

otherwise. The information flow Γ *is restricted accordingly. On the other hand,*

$$\mathbf{C} = \mathbf{A} \cup (\mathbf{A}' - \mathbf{B}') \tag{3-12}$$

COMMENT 6: The only difference between process synchronization and process communication lies in the weakened constraint (3-11), compared to (3-6). Notice that $W \subset V$ holds. Notice also that process synchronisation is associative, but generally not commutative: in $P' \parallel_{synchro} P$, we would take $\mathbf{C} = \mathbf{A} \cup (\mathbf{A}' - \mathbf{B}')$, which is different from (3-12), since non boolean ports with identical names do not carry the same values, and thus cannot be identified; hence (3-12) corresponds to a nonsymmetric choice of remaining ports. The name of process «synchronization» is due to the fact that the operation $\parallel_{synchro}$ preserves all the information needed to synchronization, namely the clocks of the signals, and the values of the boolean signals which can create new clocks through the operation of filtering. We loose apparently the ability to keep track of the multiplexing operations, since we don't care of the ceiling functions; but this lack can be overcome, as we shall see later.

3.4.2 Definition of the clock calculus.

Definition 17: Let $P = \{\Omega, \Pi, \mathbf{A}\}$ *be a process.*

(i) We shall say that $Q = \{W, \Gamma, \mathbf{B}\}$ *is an* ***extension*** *of P if there is a one-to-one mapping from* **A** *onto* **B** *which associates the ports with identical names, and*

$$\begin{array}{c} \Omega \subset W \\ \Pi_t = \Gamma_{t|\Omega} \\ \{\mathbf{A} \ni [a, X, H] \leftrightarrow [a, \hat{X}, \hat{H}] \in \mathbf{B}\} \Rightarrow \begin{cases} X = \hat{X}_{|\Omega} \\ H = \hat{H}_{|\Omega} \end{cases} \end{array} \tag{3-13}$$

(ii) The ***clock calculus*** *of* $\underset{\wedge}{P}$ *is the smallest extension of P satisfying the following property: for every non-boolean signal* $\hat{X}$ *with values in the set* Ξ,

$$\exists w : \hat{H}_t(w) < \infty \Rightarrow \{\hat{X}_t(w) : w \in W\} = \Xi \tag{3-14}$$

The clock calculus of P will be denoted by ***clock(P).***

In other words, the clock calculus of P is obtained by removing the bounds on the values of the non-boolean signals. If the set of the extensions of P satisfying (3-14) is non void, then a smallest extension clearly does exist. A sufficient condition for the existence of a clock calculus is given now.

Theorem 4: *The set of the processes which admit a clock calculus is a «process algebra», i.e. is stable under process communication.*

Proof: an immediate consequence of the formula

$$clock(P \parallel Q) = clock(P) \parallel_{synchro} clock(Q) \tag{3-15}$$

Notice that $\parallel_{synchro}$ is commutative when applied to clock calculi, since no bound does exist on the non boolean ports in this case, so that non boolean ports with identical names can be identified. Consequently, it suffices to know the clock calculus of a system of generators of a process algebra to know the clock calculus of any of its elements.

3.5 The algebraic clock calculus.

Ouu purpose is to show that there exists an elegant algebraic representation of clock calculi.

3.5.1 Preliminaries.

Definition 18: *We shall say that a process* $P = \{\Omega, \Pi, \mathbf{A}\}$ *is* ***totally observed*** *if, according to (3-3),*

(i): we have

$$\Pi = \sigma(\mathbf{A}) \tag{3-16}$$

(ii): furthermore all the elements of **A** *possess a clock taking its values in the set* **N**.

In other words, we assume

1. that no clock nor any signal which essentially contributes to the information flow has been masked
2. that the basic clock *Id* dominates all the clocks of elements of **A** in the sense of (2-19).

COMMENT 7: For totally observed processes, observing the set of the signals implies that the whole history is observed. Moreover, no multiplexing takes place in the clocks of the ports of this process. This restriction implies that we shall not be able to handle the multiplexing operation on clocks; but it turns out that the multiplexing can be rebuilt using the instructions of the mini-language SL_4, as we have shown before, and, on the other hand, we shall see later that the algebra of processes generated by the instructions of SL_4 satisfies the conditions above.

Algebraic clock calculi will be defined for totally observed processes only.

Now consider a process $P = \{\Omega, \Pi, \mathbf{A}\}$ and its clock calculus $clock(P) = \{W, \Gamma, \mathbf{B}\}$. We introduce on W the following family of equivalence relations

$$w \approx_{clock(s)} w'$$

$$\Leftrightarrow$$

$$\begin{cases} \forall [b, X, H] \in \mathbf{B} \ X \text{ boolean}: \ \forall t \leqq s \ X_t(w) = X_t(w') \text{ and } H_t(w) = H_t(w') \\ \forall [z, Y, K] \in \mathbf{B} \ Y \text{ non boolean}: \ \forall t \leqq s \ K_t(w) = K_t(w') \end{cases} \tag{3-17}$$

We shall denote by

$$algclock(P) := \{\Xi, \Delta, \mathbf{B}\} \tag{3-18}$$

the ***algebraic clock calculus*** of P, which is defined as follows

$$\Xi = \text{quotient space of } W \text{ w.r.t. } \approx_{clock(\infty)}$$

$$\Delta_s: \ \xi \approx_s \xi' \Leftrightarrow \forall w \in \xi, w' \in \xi': \ w \approx_{clock(s)} w'$$

$$\mathbf{B} \text{ unchanged} \tag{3-19}$$

The algebraic clock calculus of P summarizes its synchronisation properties. We shall now show that there is a very elegant representation of *algclock* (P).

3.5.2 The fundamental map.

We consider again the processes P and *algclock* (P) defined above. In the sequel, Z/3Z will denote the finite field with 3 elements $\{-1,0,+1\}$. We consider on the other hand the labels $\{absent, true, false, present\}$; these labels will be encoded into Z/3Z as follows:

$$\begin{aligned} absent &\rightarrow 0 \\ true &\rightarrow +1 \\ false &\rightarrow -1 \\ present &\rightarrow \pm 1 \end{aligned} \tag{3-20}$$

where ± 1 denotes a nondeterminate choice of anyone of the values $+1$ or -1. This assignment will be used in the sequel.

The field Z/3Z will be used as follows: for $[b,X,H] \in \mathbf{B}$, set

$$\text{if} \quad b \text{ is boolean}: \beta_t(\xi) = \begin{cases} X_s(\xi) & \text{if } H_s(\xi) = t \\ absent & \text{if } t \notin \overline{H}(\xi,.) \end{cases}$$

$$\text{if} \quad b \text{ is non boolean}: \beta_t(\xi) = \begin{cases} present & \text{if } t \in \overline{H}(\xi,.) \\ absent & \text{if } t \notin \overline{H}(\xi,.) \end{cases}$$

and denote by

$$[b, \beta, Id] = \varphi_{clock}([b,X,H]) \tag{3-21}$$

the so-defined map. Using these notations, we have the following fundamental theorem.

Theorem 5: *The map* Φ_{clock} *defined by*

$$\xi \rightarrow \{\beta(\xi)\}_{[b,X,H] \in \mathbf{B}} \tag{3-22}$$

where β *is given by (3-21), is an isomorphism between the process algclock(P) and the process* $\{\mathbf{V}, o(\mathsf{B}), \mathsf{B}\}$ *where*

$$\mathbf{V} = \Phi_{clock}(\Xi) \subset \{\mathbf{N} \rightarrow \mathbf{Z}/3\mathbf{Z}^{\#\mathbf{B}}\}$$

$$\mathsf{B} = \varphi_{clock}(\mathbf{B})$$

where # denotes as usually the cardinal. Points of $\mathbf{V}$ *will be generically denoted by* v .

Proof: The map Φ_{clock} is clearly injective. The isomorphism between the involved histories is a direct consequence of the fact that the original process P was assumed totally observed, which implies the same property for the process *algclock* (P).

In other words, theorem 5 expresses that the clock calculus of a process is canonically represented by some submanifold of the space $\{\mathbf{N} \to \mathbf{Z}/3\mathbf{Z}^{\#\mathbf{B}}\}$. This rather abstract and non-effective result can nevertheless be used to further characterize the properties of starvation and temporal nondeterminism we have introduced before.

Remark: Recall that process communication requires as a preliminary that a nondeterminate amount of dummy events could be inserted between two successive events of each sub-process. This operation is rather simple for algebraic clock calculi, since it is equivalent to allow all the signals to be zero simultaneously. In the sequel, *the clock calculus of a given process will implicitely refer to any such extension.*

3.5.3 Starvation, temporal nondeterminism, and clock calculus.

Consider again the algebraic clock calculus $\{\mathbf{V}, o(\mathsf{B}), \mathsf{B}\}$ where the elements of B are denoted by $[b, \beta, Id]$, and recall that $\mathbf{V}$ is a submanifold of $\{\mathbf{N} \to \mathbf{Z}/3\mathbf{Z}^{\#\mathbf{B}}\}$. Recall that elements of $\mathbf{V}$ are denoted by ν.

3.5.3.1 Starvation and death time.

Let us pick a port $[b, \beta, Id]$, and define its *life time* as being the function

$$\lambda_\beta(\nu) = \max\{t : \beta_t(\nu) \neq 0\} \tag{3-23}$$

Notice that this function does not satisfy the causality condition (2-7) of the definition of clocks: it might happen that $\lambda_\beta(\nu) \leqq t$ while being unable to recognize this fact at time t (cf. the counterexamples 4 following the definition of clocks). Recall this causality condition to be satisfied for an $\mathbf{N}$-valued function τ we shall call an *event:*

$$\tau(\nu) \leqq t \text{ and } \nu' \approx_t \nu \Rightarrow \tau(\nu) = \tau(\nu') \tag{3-24}$$

Then, the *death time* of the port $[b, \beta, Id]$ is defined as follows

$$\delta_\beta = \min\{\tau \text{ event}, \ \tau \geqq \lambda_\beta\} \tag{3-25}$$

where λ_β is the life time of β. Hence the death time of a port is the first time from which on this port surely starvates for ever.

Examples.

Consider the *SL* program

```
y = x when b
```

Then, the life time of y might be finite: this happens if after some time t_0 the boolean b fails to be true; but the future of the signal b is not known, so that we don't know at time t_0 that y will be absent for ever. The death time of y is actually infinite!

On the other hand, consider the program

```
b = (x<z)
||
y = x when b
||
u = y+z
```

This program possesses two inputs, namely x and z, the values of which should be considered as unconstrained. But this program exhibits a contradiction: x, b and z must have the same clock by the first instruction; then u, y and z must also have the same clock by the last instruction; but y is less frequent than x and b by the second instruction, since one cannot prevent from (x<z)=*false* to occur. This program starvates from the beginning: the death time of all the ports is 0.

These examples illustrate that the subtle notion of death time is properly related to starvation, while the more obvious notion of life time is not. We shall indicate in the sequel effective algorithms to calculate death times, i.e. to predict starvation of a program in a static way, without running it.

3.5.3.2 Temporal nondeterminism.

As we have indicated before, temporal nondeterminism can occur if the process is observed via a subset C of ports (for instance a set of «input ports», or a set of «non masked ports»). We shall give in the sequel effective algorithms to recognize temporal nondeterminism via the clock calculus.

Chapter Four
A DENOTATIONAL SEMANTICS FOR THE *SL* LANGUAGES.

The purpose of this chapter is to illustrate the power of the model of processes on the family of the *SL*-languages.

4.1 Notations.

4.1.1 The semantic function.

The semantic function will be denoted as follows

$$\mathbf{M}[\text{program}] \;::\; \left(\begin{array}{c} \mathbf{EPROC}[\text{program}] \\ \hline \mathbf{SYNCH}[\text{program}] \\ \hline \mathbf{VAL}[\text{program}] \\ \hline \mathbf{ALGCLOCK}[\text{program}] \end{array}\right)$$

where

- «program» denotes a SL_4-program.
- **M[ ]** denotes the semantic function, the result of which is a process in the sense of the definition 13; the process **M[program]** is entirely defined by the first three fields specified on the right handside.
- the field **EPROC** denotes a process which is an extension of **M[program]** (definition 17−i); the convenient restriction is specified by the two next fields.
- the field **SYNCH** is a specification of the relations between the clocks of the ports of **M[program]**.
- the field **VAL** is a specification of the constraints on the various signals involved in **M[program]**.
- the additional field **ALGCLOCK** is a specification of the algebraic clock calculus of **M[program]**.

To summarize, the style of the semantics is the following. First, we specify an extension of the desired process; second we specify the convenient restriction of the space Ω of this extension, and this is obtained via the specification of constraints on signals and clocks of the ports involved in the extended process.

4.1.2 Notations for the field «EPROC»

4.1.2.1 Canonical process associated to a signal.

Given a signal x of type Ξ involved in a program, we introduce

$$\mathbf{M}[\mathrm{x}] = \{\Omega^{\Xi},\ o([\mathrm{x}, X, Id]),\ [\mathrm{x}, X, Id]\}$$

$$\Omega^{\Xi} = \{\mathbf{N} \to \Xi\}$$

$$\omega \in \Omega^{\Xi}:\ X_t(\omega) = \omega(t) \qquad (4\text{–}1)$$

M[x] is a process we shall call the *canonical process of* x. This process summarizes what can be known when observing x only.

4.1.2.2 Canonical process associated to a variable.

Variables are required in the «delays» to specify initial conditions. If u is a variable of type Ξ, we introduce the *canonical process of the variable* u as being

$$\mathbf{M}[\mathrm{u}] = \{\Xi,\ o([\mathrm{u}, Id, 0]),\ [\mathrm{u}, Id, 0]\} \qquad (4\text{–}2)$$

where 0 denotes the clock H such that $H_0 = 0, H_t = \infty$ for $t > 0$, i.e. the clock which has the initial instant as only instant. The information flow on Ξ is just the constant partition the atoms of which are the points of Ξ.

When u is a variable, and x is a signal, it is easy to show that the process **M[u]** $\|$ **M[x]** summarizes what can be known when observing first u as initial condition, and then the signal x. To refer to this interpretation, we shall use the notation of concatenation, and denote this process simply by

$$\mathbf{M}[\mathrm{u}].\mathbf{M}[\mathrm{x}] \qquad (4\text{–}3)$$

4.1.2.3 Use of process communication.

We shall also make use of the process communication P $\|$ Q as defined in the definition 14.

4.1.3 Notations for the field «SYNCH».

Relationships between clocks will be simply specified by a set of relations written in terms of the operations of the clock algebra, namely the four operations

$$\vee,\ \wedge,\ \downarrow,\ \uparrow$$

respectively defined in (2–20), (2–22), (2–27). Given two such sets **SYNCH1** and **SYNCH2**, we shall denote by **SYNCH1** ∪ **SYNCH2** the union of these sets, where identical names refer to the same objects. Finally, given a port $[b, X, H]$, we shall often write generically $H(b)$ to denote the clock of the port named b.

4.1.4 Notations for the field «VAL».

We have to specify relationships between the values of different signals at every instant. Such relations will be specified in the following way. Given two ports $[b, X, H]$ and $[b', X', H']$, we shall write formulas of the form

$$K: f(X, X') \tag{4–4}$$

In this formula

- K denotes a clock, or an expression written in terms of the clock algebra, satisfying the condition $K \subset H \wedge H'$.
- $f(x, y)$ denotes a relation.

The meaning of (4–4) is the following (cf. (2–9) for the definition of traces):

$$\forall(\omega, \theta) \in \overline{K}: \begin{cases} H_t(\omega) = \theta \\ H'_s(\omega) = \theta \end{cases}$$

$$\Downarrow$$

$$f(X_t(\omega), X'_s(\omega)) \tag{4–5}$$

A finite set of such formulas will be sufficient to specify entirely the desired constraints on the values of signals. As for the preceding field, the notation **VAL1** ∪ **VAL2** denote the union of the referred sets of relations on values. Finally, given a port named b, we shall generically denote by the same name «b» the signal of this port.

4.1.5 Notations for the field «ALGCLOCK».

As we have shown before, the algebraic calculus of a process is defined via the specification of a subset of the set of all the possible trajectories of a point moving in the product $\mathbf{Z/3Z}^n$ for some integer n. In the present case, it will suffices to work with *dynamical systems* (also called *iterations* in the field of algebraic geometry) over $\mathbf{Z/3Z}^n$.

Definition 19: A dynamical system over $\mathbf{Z/3Z}^n$ *is the specification of*

1. *a submanifold of the product space* $\mathbf{Z/3Z}^n$ multpl $\mathbf{Z/3Z}^n$;
2. *an initial condition in* $\mathbf{Z/3Z}^n$.

Indeed, denoting by ξ the generic point of $\mathbf{Z/3Z}^n$, such a submanifold is specified via a system of polynomial equations

$$\begin{aligned} P_1(\xi, \xi') &= 0 \\ &\cdots\cdots \\ &\cdots\cdots \\ P_k(\xi, \xi') &= 0 \end{aligned} \tag{4-6}$$

where $\xi = (x_1, \ldots, x_n)$, and the x_i's are variables in Z/3Z. Then, *the dynamical system is the subset of the trajectories on* $\mathbf{Z/3Z}^n$ *satisfying*

$$\begin{aligned} P_1(\xi_t, \xi_{t-1}) &= 0 \\ &\cdots\cdots \\ &\cdots\cdots \\ P_k(\xi_t, \xi_{t-1}) &= 0 \end{aligned} \tag{4-7}$$

where ξ_0 equals the given initial condition.

Given two such dynamical systems **ALGCLOCK1** and **ALGCLOCK2**, the notation **ALGCLOCK1** ∪ **ALGCLOCK2** denotes the dynamical system specified by the union of the two sets of algebraic equations and initial conditions.

We are now ready to present the semantics of SL_4 and of the instruction «multiplexing» of SL_3.

4.2 The semantics of *SL*.

4.2.1 Instruction (o): relation.

4.2.1.1 Non boolean relation.

$$\mathbf{M}[\mathrm{p}(\mathrm{x}(1),\ldots,\mathrm{x}(\mathrm{n}))] \;::\; \left(\begin{array}{c} \mathbf{M}[\mathrm{x}(1)] \,\|\, \ldots. \,\|\, \mathbf{M}[\mathrm{x}(\mathrm{n})] \\ \hline H(\mathrm{x}(1)) = \ldots = H(\mathrm{x}(\mathrm{n})) \\ \hline H(\mathrm{x}(\mathrm{i})):\ p(x(1),\ldots,x(n)) \\ \hline x(1)^2 = \ldots = x(n)^2 \end{array}\right)$$

We choosed as extended process the communication of the canonical processes of the involved ports; in this case, communication is equivalent to juxtaposition. All signals have the same clock. When the signals are present, their values must satisfy the specified relation. The algebraic calculus is just the algebraic coding of the relations between clocks. Notice that, in this framework, we consider the instruction

b = (x<y)

the result of which is a boolean signal, as a non boolean relation, so that the corresponding algebraic clock calculus yields $b^2 = x^2 = y^2$, i.e. does not determine the value of b.

4.2.1.2 Boolean relation.

If f(x(1),...x(n)) is a boolean relation, the first three fields of its semantics are again given as above. The only change lies in the algebraic clock calculus, since boolean values are taken into account in this case. Let us first list the algebraic clock calculi of elementary boolean relations:

$$\begin{aligned} \mathbf{ALGCLOCK}[\mathrm{a} = \mathit{true}] &= \{a^2 - a = 0\} \\ \mathbf{ALGCLOCK}[\mathrm{b} = \text{not a}] &= \{b = -a\} \\ \mathbf{ALGCLOCK}[\mathrm{x} = \mathrm{a} \text{ and } \mathrm{b}] &= \left\{\begin{array}{c} a^2 = b^2 \\ x = a^2 - (ab + a + b) \end{array}\right\} \end{aligned} \qquad (4\text{–}8)$$

The first equation means that a must be absent or *true*, and so on. Using these formulas, the algebraic clock calculus of any boolean relation is easily built by rewriting.

4.2.2 Instruction (i): the register.

4.2.2.1 Non boolean register.

$$\mathbf{M}[y = u \rightarrow \$x] \quad :: \quad \left(\begin{array}{c} \mathbf{M}[\mathrm{u}] \ . \ \mathbf{M}[\mathrm{x}] \\ \hline H(\mathrm{y}) = H(\mathrm{x}) \\ \hline H(\mathrm{x}) : y_1 = u; \forall t > 1 : y_t = x_{t-1} \\ \hline y^2 = x^2 \end{array} \right)$$

To define the extended process, we used the notation (4–3). Notice that, for the sake of clarity, we did not mention in the specification of this extended process the output port named y. This is of no importance since this port has no influence on the definition of the history of this extended process: the output signal y is just a function of the input signal x and of the initial condition u, as indicated in the other fields of the semantics. The same convention will be used in the sequel, without any further reference. On the other hand, the algebraic clock calculus just expresses that x and y possess the same clock.

4.2.2.2 Boolean register.

The only difference with the non boolean register lies in the last field, namely the algebraic clock calculus, since boolean values are considered in clock calculi. In this case, we have

$$\mathbf{ALGCLOCK}[y = u \rightarrow \$x] = \left\{ \begin{array}{c} \xi = (1 - x^2)\xi' + x \\ y = x^2 \xi' \\ \text{initial condition} : \xi'_0 = u \end{array} \right\} \qquad (4\text{–}9)$$

and the dynamical system is defined by the transformation

$$\begin{pmatrix} \xi' \\ x' \\ y' \end{pmatrix} \rightarrow \begin{pmatrix} \xi \\ x \\ y \end{pmatrix}$$

$$\text{initial condition} = \begin{pmatrix} u \\ \textit{arbitrary} \\ \textit{arbitrary} \end{pmatrix} \qquad (4\text{–}10)$$

where $((\xi', x', y'), (\xi, x, y))$ are related via (4–9). The additional component ξ of the so defined dynamical system represents the history of the internal memory of the register; it is clear that $\xi \neq 0$ holds, which expresses the fact that a value is always present in the memory. The meaning of the dynamical system (4–9) is the following: the internal memory of the register remains unchanged between two successive occurrences of x ($x = 0$ at such an instant), while the value of x is fed in this memory when x is present ($\xi = x$ at such instants); at the same time, the past content of the memory is delivered at the output y ($y = \xi'$).

4.2.3 Instruction (ii): the condition.

4.2.3.1 Condition with non boolean output.

$$\mathbf{M[y = x\ when\ b]} \quad :: \quad \left(\begin{array}{c} \mathbf{M[b] \parallel M[x]} \\ \hline H(\mathrm{y}) = H(\mathrm{x}) \wedge (H(\mathrm{b}) \downarrow \mathrm{b}) \\ \hline H(\mathrm{y}) : y = x \\ \hline\hline y^2 = x^2(-b - b^2) \end{array} \right)$$

The notation $\downarrow$ refers to the filtering of a clock by a test, see (2–22). The third field expresses that, if y is present, it carries the value of x. The algebraic clock calculus is equivalent to the constraint specified in the field **SYNCH**.

4.2.3.2 Condition with boolean output.

Again, the only difference lies in the algebraic clock calculus. Since the value of boolean signals is taken into account, we must have

$$\mathbf{ALGCLOCK[y = x\ when\ b]} = \{y = x(-b - b^2)\} \qquad (4\text{–}11)$$

4.2.4 Instruction (iii): the multiplexing.

$$\mathbf{M}[\mathrm{h} = \mathbf{mux}\,\mathrm{c}] \;::\; \left(\begin{array}{c} \mathbf{M}[\mathrm{c}] \\ \hline H(\mathrm{h}) = H(\mathrm{c}) \uparrow \mathrm{c} \\ \hline H(\mathrm{h}) : h = \mathit{true} \\ \hline \varnothing \end{array}\right)$$

We cannot define an algebraic clock calculus for the multiplexing, since it is not a totally observed process, see definition 18. On the other hand, we should keep in mind that the simultaneous use of the instructions **mux** and **default** can cause trouble and is in fact not well defined, as it has been pointed out in the second fundamental remark following theorem 2. Nevertheless, *the language SL_3 is well defined, since no difficulty results from the simultaneous use of the operations $\downarrow$, $\uparrow$ and $\wedge$ on clocks.*

4.2.5 Instruction (iv): the merge.

4.2.5.1 Merge with non boolean output.

$$\mathbf{M}[\mathrm{y} = \mathrm{u}\,\mathbf{default}\,\mathrm{v}] \;::\; \left(\begin{array}{c} \mathbf{M}[\mathrm{u}] \parallel \mathbf{M}[\mathrm{v}] \\ \hline H(\mathrm{y}) = H(\mathrm{u}) \vee H(\mathrm{v}) \\ \hline H(\mathrm{u}) : y = u;\;\; [H(\mathrm{v}) - (H(\mathrm{u})) \wedge H(\mathrm{v}))] : y = v \\ \hline y^2 = u^2 + v^2(1 - u^2) \end{array}\right)$$

4.2.5.2 Merge with boolean output.

The algebraic clock calculus is modified as follows:

$$y = u + v(1 - u^2) \tag{4-12}$$

4.2.6 Instruction (v): communication.

$$M[P \parallel Q] :: \left(\begin{array}{c} M[P] \parallel M[Q] \\ \hline SYNCH[P] \cup SYNCH[Q] \\ \hline VAL[P] \cup VAL[Q] \\ \hline ALGCLOCK[P] \cup ALGCLOCK[Q] \end{array} \right)$$

In fact, only the first field is necessary to give the entire semantics, thanks to the definition 14 of the process communication. On the other hand, the last field is equivalent to the formula (3–15). We have nevertheless written all the fields explicitely for the sake of clarity.

Another equivalent semantics of the composition is the following:

$$M[P \parallel Q] :: \left(\begin{array}{c} EPROC[P] \parallel EPROC[Q] \\ \hline SYNCH[P] \cup SYNCH[Q] \\ \hline VAL[P] \cup VAL[Q] \\ \hline ALGCLOCK[P] \cup ALGCLOCK[Q] \end{array} \right)$$

Here, **EPROC** denotes *any extension of the mentioned process.* In this case, the specification of the other fields is necessary. The latter semantics will be used in the proof of the next section.

4.3 Applications.

4.3.1 Proof that SL_3 is included in SL_4.

Consider again the SL_4-program of the multiplexing:

multiplexing =

```
(i1)    u = zu − 1
        ||
(i2)    zu = raz default past_u
        ||
(i3)    past_u = u_0 → $u

        ||

(i4)    raz = C when past_stop
        ||
(i5)    past_stop = true → $stop
        ||
(i6)    stop = (u = 0)

        ||

(i7)    synchro C, raz
```

The semantics of this program is written as follows.

EPROC[multiplexing]=
M[zu] || M[raz] || M[past_u] || M[u] . M[u] ||
M[past_stop] || M[stop] || M[u] || M[C] ||

SYNCH[multiplexing]=
H(u)=H(zu)
H(zu)= H(raz) ∨ H(past_u)
H(past_u)=H(u)
H(raz)=H(C) ↓ past_stop
H(past_stop)=H(stop)
H(stop)=H(u)
H(C)=H(raz)

VAL[multiplexing]=
H(u): u=zu−1
H(raz): zu=raz; H(past_u) − H(raz): zu=past_u
H(u): past_u= u_0 → $u
H(raz): raz=C
H(stop): past_stop= *true* → $stop
H(u): stop=(u=0)

In this field, the notation «past_u= u_0 → $u» stands for short instead of the explicit specification of the field **VAL** of the register.

Using immediate properties of the clock algebra, the field **SYNCH** can be rewritten as

SYNCH[multiplexing]=
H(u)=H(zu)=H(past_u)=H(stop)=H(past_stop)
H(C)=H(raz)=past_stop

As a consequence, the field **VAL** is rewritten as

VAL[multiplexing]=

H(u):
u=zu−1
past_u= u_0 → $u
past_stop= *true* → $stop
stop=(u=0)

past_stop (=H(C)):
raz=C
zu=raz

H(u) − past_stop:
zu=past_u

Let us now prove that the process **M[h = mux C]** can be embedded in **M[multiplexing]**. To simplify the notations, we shall write for short H to denote the clock H(C), and T to denote the boolean signal past_stop. On the other hand, we shall denote by ω the points of the trajectory space of **M[multiplexing]**. Finally, notice that it is always possible to choose the fastest clock as the identity clock: here, this yields H(u)=Id. We are now ready to present the proof.

STEP 1. Choose an ω and an integer s. Consider the set of the integers t such that

$$H_s(\omega) \leqq t \leqq H_{s+1}(\omega)$$

For such t's, it is easy to derive from **VAL[multiplexing]** the following equalities

$$\begin{aligned} & T_t(\omega) = (u_{t-1}(\omega) = 0) \\ t < H_{s+1}(\omega) \;\Rightarrow\; & u_t(\omega) = C - (t - H_s(\omega)) \end{aligned} \tag{4-13}$$

which imply

$$T_t(\omega) = \mathit{true} \;\Leftrightarrow\; t = H_s(\omega) + C_s(\omega) + 1 = H_{s+1}(\omega)$$

On the other hand, it is clear form (4-13) that, stricly before time $H_{s+1}(\omega)$, every signal

depends only on the initial segment of C up to and including its s−th occurrence. Finally, we have proved

$$\begin{gathered} H = Id \downarrow T \\ H_{s+1}(\omega) = H_s(\omega) + C_s(\omega) + 1 \\ \left.\begin{array}{c} t < H_{s+1}(\omega) \\ \omega' \approx_t \omega \end{array}\right\} \Rightarrow \omega' \approx_{H_s} \omega \end{gathered} \tag{4-14}$$

Now, define on Ω the information flow (cf. definition 4)

$$\Gamma_s = \Pi_{H_s} \tag{4-15}$$

Then, it is clear from (4-14) and (4-15) that

1. C is a $\{\Omega, \Gamma, Id\}$-signal,
2. T, u are signals with clock $Id \uparrow C$

This finishes the proof of the theorem. □

4.3.2 The algebraic clock calculus as a proof system.

Let us begin with the following result.

Lemma 6:

The general form of an algebraic clock calculus of an *SL*-process is the following

$$\begin{gathered} \xi(i) = (1 - x(i)^2).\xi'(i) + x(i), \quad i = 1, \ldots, k \\ y(i) = x(i)^2 \xi'(i), \quad \text{initial condition } \xi(i)_0 \\ C_j(x(1), \ldots, x(k), y(1), \ldots y(k), u(1), \ldots u(m)) = 0, \quad j = 1, \ldots, p \end{gathered} \tag{4-16}$$

where the C_j's are polynomial of degree almost 2 with respect to each of the involved variables, satisfying the property

$$C_j(0, \ldots, 0) = 0 \ \square \tag{4-17}$$

According to the formula (4-6), (4-16) defines an algebraic clock calculus. Before proving this result, let us comment what are the objects we have introduced.

COMMENT 8: The pairs $\xi(i), \xi'(i)$ represents the new and old state of the register memories, the initial state of which are specified in $\xi(i)_0$. The components of x, y, u are the signals of the considered process; recall that only the signals are visible from the external world. Notice also

that the trajectory of the internal state ξ is entirely defined by the trajectories of the components of x, y, u: *SL*-processes are always deterministic for an observer showing all the signals. The dynamical part of (4–16) expresses that the memory state changes only when signals are present. The property (4–17) implies that the absence of all the signals (i.e. the total starvation of the process) is always an admissible trajectory of the algebraic clock calculus.

Proof: an easy consequence of the algebraic clock calculi of the *SL*-instructions; notice that the special form of the dynamics is a direct consequence of the algebraic clock calculus of the boolean register, see (4–9).

Lemma 7: Consider again the same process P as before. We shall denote by **STATCLOCK[*P*]** the *static clock calculus of P*, which is the set of all the values taken by the trajectory of x during the life of the dynamical system, for all possible initial conditions. Then, **STATCLOCK[*P*]** is built using the same rules as for **ALGCLOCK[*P*]**, except for the equation of boolean register, which is replaced by the corresponding equation of non boolean register.

The proof is immediate, and is left to the reader.

4.3.2.1 Determinism.

Theorem 7: *Let* $P = \{\Omega, \Pi, \mathbf{A}\}$ *be a totally observed* (definition 18) *process, and let* $\mathbf{C} \subset \mathbf{A}$ *be a subset of ports of P* (**C** is intended to refer to a subset of «non-masked» ports of P), and denote by **B** the subset of **A** composed by the boolean ports involved in a non boolean relation. *Then,* $(i) \Rightarrow (ii)$, *where*

(i): this process is deterministic with respect to the observer $o(\mathbf{C})$

(ii): its static clock calculus satisfies the following condition (cf. lemma 7)

(DET)*: every point in* **STATCLOCK[*P*]** *is entirely determined by its components in* $\mathbf{C} \cup \mathbf{B}$.

COMMENT 9: the theorem expresses that nondeterminism in a process can be detected using the clock calculus.

Proof: easy, left to the reader; it is not possible to cancel the set **B**, since, although the values of the corresponding signals are well defined by the observation of C in a deterministic process, these values can be unconstrained in **STATCLOCK[*P*]**, as the example given after the semantics of the non boolean relation shows.

To illustrate the power of this criterion, we shall use it to analyse the program of the multiplexing. Our objective is to prove in this way that this program cannot be rejected as nondeterministic when the input «ceiling function» C is taken as an observer. Consider first the instructions (i1–i3) of this program. The corresponding static (as well as algebraic in the present case) clock calculus is given by

$$u^2 = zu^2 = past\_u^2$$
$$zu^2 = raz^2 + past\_u^2(1 - raz^2)$$

the second equation of which is equivalent to

$$raz^2(1 - u^2) = 0 \qquad (4\text{-}18)$$

This relation expresses that the signal u must be present when raz is present. If we choose the input signal of this program, namely raz, as an observer, (4-18) shows that the clock of u is not determined by the clock of raz. This program does not satisfy the condition (DET). Indeed, it is easily recognized as nondeterministic.

Let us now write the static clock calculus of the entire program:

$$u^2 = zu^2 = past\_u^2 = stop^2 = past\_stop^2$$
$$zu^2 = raz^2 + past\_u^2(1 - raz^2)$$
$$C^2 = raz^2 = -past\_stop - past\_stop^2$$

It is easy to see that the first and last equations imply the second one, so that we simply get

$$u^2 = zu^2 = past\_u^2 = stop^2 = past\_stop^2$$
$$C^2 = raz^2 = -past\_stop - past\_stop^2$$

Since $past\_stop$ belongs to the set **B** introduced in the theorem 7, it is clear that the above process satisfies (DET) with C as only observer. This is satisfactory, since we have already proven that this program is isomorphic to the multiplexing, so that it is indeed deterministic with C as only observer.

4.3.2.2 Starvation.

It is unfortunately not possible to build a complete set of criteria to check starvation within the present relational model. As the investigation of the forthcoming examples will show, starvation must be analysed in close relation with control and data dependencies, which are notions we do not take into account in the present model. See [Le Guernic & Benveniste 1986] for related effective criteria for SIGNAL programs, and [Boussinot 1986] for ESTEREL programs.

To illustrate our purpose, notice first that the investigation of the static clock calculus of the multiplexing suffices to show that this program never starvates. Now, consider the incorrect program we have introduced before, namely

```
b = (x<z)
||
y = x when b
||
u = y+z
```

Its static clock calculus is

$$b^2 = x^2 = z^2 = u^2 = y^2$$
$$y^2 = x^2(-b - b^2)$$

which implies $b^2 - b = 0$. An interpretation of this result is needed. While the clock of the boolean signal b can be accepted as constrainted, this cannot be the case for the value of b. Hence this constraint *must be solved for* b^2, which yields in this case the desired result, namely $b \equiv 0$: the program starvates from the beginning. A careful investigation shows that the key point is that *the value of* b *cannot be influenced by the clocks of the signals this value depends upon*; here these signals are x and z, but the **when** instruction requires such an influence.

4.3.2.3 Some hints for effective algorithms.

As the reader may have guess, the kind of reasoning we have performed with clock calculi uses algebraic elimination techniques as a primary tool. While general techniques of elimination theory are based on the construction of standard bases of polynomials ([Buchberger 1970, 1979]), simpler techniques can be used in our case taking into account that we are working in the manifold $\mathbf{Z}/3\mathbf{Z}^n$, which is of algebraic dimension 0; elimination can be performed using the explicit formulas for algebraic equations of degree 2. Such an algorithm is outlined in [Le Guernic & Benveniste 1986].

4.4 Conclusion of this chapter.

The main conclusion we derive from the study of the semaintcs of SL_4 is that the multiplexing (the importance of which has been already discussed) can be simulated in the language SL_4, which does not use this instruction as a primitive. This is a rather important fact, since the multiplexing is a cause of trouble in the calculation of the synchronisation of processes, while the language SL_4 does not suffer from this drawback.

Chapter Five
CONCLUSION AND DISCUSSION.

5.1 Conclusion: Ω-calculus.

- We have presented a model of histories and more specific model of processes. For obvious reasons, the whole technique will be referred to as «Ω-calculus». The style of these models is denotational, but the fully functional viewpoint ot handle events allowed us to dispense with the technique of continuation to construct process communication.
- This model allowed us to study deeply the notion of synchronisation, and the process algebras generated by process communication. Starting from the principles of synchrony, we have shown how nondeterminism and starvation can result from process communication. A major contribution is that our point of view lead to the algebraic clock calculus, which is an effective tool to recognize nondeterminism and starvation (although the latter point would need a refined model).

5.2 About the synchronous languages.

As we have mentioned before, we know at least four languages based on the principles of synchrony: LUCID [Ashcroft & Wadge 1976], ESTEREL [Berry & Cosserat 1984], [Boussinot 1986], LUSTRE [Bergerand & al. 1985], and SIGNAL [Le Guernic & al. 1986], [Le Guernic & Benveniste 1986].

SIGNAL is tightly connected to the mini-language SL_4. SIGNAL is a functional language, hence its compiler calculates the synchronisation *and the data/control dependencies* in a program; SIGNAL is based on a less general, but finer model of synchronous programming which is presented in an operational style in [Le Guernic & Benveniste 1986]. The first version of SIGNAL calculates the synchronisation of a program with the aid of the calculus of the algebra of clocks ($\vee$, $\wedge$, $\downarrow$), except for the multiplexing. A second version, in preparation, implements the algebraic calculus we have presented here, using elementary elimination techniques of algebraic geometry; this second version is able to reason at the same time on booleans and on clocks. An extended version, also in preparation, implements the dynamical clock calculus.

LUSTRE is a slight modification of SL_2. The modification is the following: in the instruction (ii) (y = x **when** b), the two input signals are required to possess the same clock. As a consequence, the set of the clocks of a LUSTRE program has the structure of a tree: no inference need to be performed on the synchronisation, which is entirely specified by the programmer. This makes LUSTRE less powerful, but simpler than SIGNAL; for example, nondeterminism cannot be introduced by the synchronisation mechanisms of LUSTRE, unlike to SIGNAL, and also ESTEREL as we shall see later.

LUCID is obtained by adding the instruction (iii) (multiplexing) to SL_1. LUCID operates on sequences, and knows nested loops. The control is performed via the nesting of loops and the instruction **as soon as**, the combination of which corresponds to our multiplexing. On the other hand, the delay $ is replaced by the operator **next** which refers to the future. The structure of the clocks of a LUCID program is a tree with the master clock (the fundamental sequence index) as root, and the successive nested loops (e.g. ceiling functions in our framework, at least when inner loops terminate) as nodes.

Finally, ESTEREL is much more difficult to relate to the present model, since it is an imperative rather than declarative language. It is our conjecture that, as far as the synchronisation mechanisms are concerned, ESTEREL is related to SL_3. The multiplexing is implicitely performed via the instruction P;Q, where ";" means that program P passes the control to program Q when it terminates: the termination of P and the firing of Q take time at the same «macro-instant», but must be considered as ordered; this point is deeply discussed in [Boussinot 1986]. The combination of the operator ; and the operator || which is the parallel communication is the cause of nondeterminism in ESTEREL, as discussed again in [Boussinot 1986]; the reader might recognize in our framework the corresponding cause of possible nondeterminism, which is the combination of || and oversampling. As in SIGNAL, effective criteria are available in ESTEREL's compiler to detect nondeterminism. However, an important difference between ESTEREL and SL_3 is the absence of ways to specify strong synchrony (our instruction (o)); the reason is that transformations on data are performed via variables which are permanent and do not possess clocks. Accordingly, the equivalent of our $, namely the instruction **pause**, do not operate on data, but only on control. As a consequence, no inference needs to be performed on clocks in ESTEREL, and non synchronization constraints can be put on the input ports, which makes ESTEREL fully «reactive».

An important consequence of the fact that SIGNAL and ESTEREL both provide tools to perform oversampling, is that *the execution scheme of ESTEREL or SIGNAL programs on a Von Neumann machine can be properly specified in the same language* (ESTEREL or SIGNAL respectively): the successive firing of all the elementary actions involved in a given instant does require the use of the multiplexing. See the final remark of Boussinot's paper about ESTEREL, and the chapter on computational semantics of [Le Guernic & Benveniste 1986] for SIGNAL.

5.3 Future directions of research.

Directions to be investigated could be the following.

- *The algebraic clock calculus as a proof system equivalent to temporal logic?* It is conjectured that {"dynamical clock calculus" + "first order logic assertions"} should possess the power of temporal logic [Pnueli 1977]. In case of a positive answer, another question would be to compare the algorithms of temporal logic with the present algebraic elimination techniques.

- *The connection with the models* CSP, SCCS, MEIJE. It is expected that the mechanisms of synchrony and asynchrony of these models could be realised in Ω-calculus. For example, it is our feeling that the algebra of *trioids* $(T, ;, \|, +, 1)$ and its set of algebraic rules, introduced in [Boudol & Castellani 1986], can be embedded in our single but rich algebraic structure of *algebraic clock calculus;* for instance, the communication with symmetric conflict $P + P'$ of Milner can be characterized in our clock calculus by constraints of the form $a.b' = 0, \Sigma a^2 + \Sigma b'^2 = 1$, where a and b' respectively range over the sets of the ports of P and P'. On the other hand, recursivity cannot be represented in our model of processes (contrary to MEIJE), and it not clear whether it can be represented in the more general model of histories.

REFERENCES.

[Ashcroft & Wadge 1976]: E.A. Ashcroft, W.W. Wadge, «LUCID – a formal system for writing and proving programs», SIAM J. Comp., vol 5 No 3, 336-354.

[Bergerand & al. 1985]: J.L. Bergerand, P. Caspi, N. Halbwachs, D. Pilaud, E. Pilaud,«Outline of a real-time Data-Flow Language», in *Real Time Systems Symposium*, San Diego Dec.1985.

[Berry & Cosserat 1984]: G. Berry, L. Cosserat, «The ESTEREL Programming Language and its Mathematical Semantics», INRIA Res. Rep. No 327, Rocquencourt, France, to appear in *Science of Computer Programming*.

[Boudol & Austry 1984]: G. Boudol, D. Austry, «Algèbre de Processus et Synchronisation», Theoretical Comp. Sc. 30, 91-131.

[Boudol & Castellani 1986]: G. Boudol, I. Castellani, «On the semantics of concurrency: partial orders and transition systems», INRIA res. rep. No 550.

[Boussinot 1986]: F. Boussinot, «Une semantique du langage ESTEREL», INRIA research rep. No 577.

[Brock & Ackerman 1981]: J.D. Brock, W.B. Ackerman, «Scenarios, a model of non determinate computation», Conf. Formal Definition of Programming Concepts, Lect. Notes on Comp. Sc. vol 107, Springer V.

[Brookes & al. 1984]: S.D. Brookes, C.A.R. Hoare, A.W. Roscoe, «A Theory of Communicating Sequential Processes» J. ACM vol 31 no 3, 560-599.

[Buchberger 1970]: B. Buchberger, «Ein algorithmisches Kriterium fur die Losbarkeit eines algebraisches Gleichungssystems, Aequat. Math. 4, 374-383.

[Buchberger 1979]: B. Buchberger, «A criterion for detecting unnecessary reductions of Grobner bases», Eurosam 79, Lect. Notes in Comp. Sc. vol 72, 3-21, Springer Verlag.

[Cardelli 1980]: L. Cardelli, «Analog Processes», Proc. of the 9th symposium on Mathematical Foundations of Computer Science, Lect. Notes on Comp. Sc. vol 88, Springer Verlag.

[Cardelli 1982]: L. Cardelli, «Real Time Agents», Proc. of the ninth Colloquium on Automata, Languages and Programming, Aarhus, Denmark, July 1982.

[Caspi & Halbwachs 1986]: P. Caspi, N. Halbwachs, «A functional model for describing and reasoning about time behaviour of computing systems», Acta Informatica 22, 595-627.

[De Bruin & Boehm 1985]: A. De Bruin, W. Boehm, «The Denotational Semantics of Dynamic Network of Processes», ACM Trans. on Prog. Lang. and Syst., vol 7 No 4, 656-679.

[Dellacherie & Meyer 1976]: C. Dellacherie, P.A. Meyer, *«Probabilites et Potentiel»*, 2nd edition, Hermann, Paris.

[Hoare 1978]: C.A.R. Hoare, «Communicating Sequential Processes», Comm. ACM 21(8), 666-678.

[Kahn 1974]: G. Kahn, «The semantics of a Simple Language for Paralle Programming» in *Proceedings IFIP 74* , J.L. Rosenfeld Ed., North Holland, Amsterdam, 471-475.

[Kahn & Mc Queen 1977]: G. Kahn, D.B. Mac Queen, «Coroutines and Network of Parallel Processes» in *Proceedings IFIP 77*, B. Gilchrist Ed., North Holland, Amsterdam, 993-998.

[Lamport 1985]: L. Lamport, «An Axiomatic Semantics of Concurrent Languages», NATO ASI Series, vol F13, Logic and Models of Concurrent Systems, K.R. Apt Ed.

[Le Guernic & Benveniste 1986]: P. Le Guernic, A. Benveniste, «Real-time synchronous data-flow programming: the language SIGNAL and its mathematical semantics», INRIA research report No 533, reprint as INRIA res. rep. No 620.

[Le Guernic & al. 1986]: P. Le Guernic, A. Benveniste, P. Bournai, T. Gautier, «SIGNAL: a Data-Flow oriented Language for Signal Processing», IEEE Trans. on ASSP, ASSP-34 No 2, 362-374.

[Milner 1980]: R. Milner, *A Calculus of Communicating Systems*, Lect. Notes in Comp. Sc. vol 92, Springer V.

[Milner 1982]: R. Milner «Four combinators for concurrency», ACM Sigacts-Sigops Symposium on Principles of Distributed Computing, Ottawa, Canada, 1982.

[Plotkin 1981]: G.D. Plotkin, «A Structural Approach to operational Semantics», Lect. Notes, Aarhus Univ.

[Pnueli 1977]: A. Pnueli, «The Temporal Logic of Programs», Proc. of the IEEE Symposium on the Foundations of Computer Science, Providence, Rhode Island.

[Young 1982]: S.J. Young, *«Real time languages: design and development»*, Elis Horwood Publishers, 1982.

Imprimé en France
par
l'Institut National de Recherche en Informatique et en Automatique